

**INFORMATION SOCIETIES TECHNOLOGY (IST) PROGRAMME**



**P2P ARCHITECT**

**“Ensuring dependability of P2P applications at architectural level”**

**DELIVERABLE EXTERNAL**

**WORKPACKAGE: WP2 - P2P Application Dependability Specification**

**D5 - Report on the dependability properties of P2P architectures (Final Version)**

**Authors: L.Melville, J.Walkerdine, I.Sommerville (Lancaster University)**

**Submission Date: 31/07/2002**

**Partners: Athens Technology Center (GR), ENGINEERING Ingegneria Informatica (IT), University of Lancaster (UK), University of Athens (GR), Siceas Services (IT), TOP PROMOTION (GR), Alpha Satellite Television (GR).**

## SUMMARY

This document represents an initial analysis of the properties that affect Peer-to-Peer architectures.

It sets out to:

- Identify the main architectures a Peer-to-Peer system can be built upon
- Identify the properties of Peer-to-Peer architectures
- Analyse each architecture with respect to the affect they can have on the identified properties

The following changes have been made since the first draft:

- Reclassifying of properties as internal, external and hybrid properties. An additional Repairability property has also been included and analysed.
- A discussion section has been added that examines how transactions can operate within P2P systems, and also discusses the possibilities for P2P software architectures
- General corrections to the text

---

<b>SUMMARY .....</b>	<b>2</b>
<b>1. INTRODUCTION.....</b>	<b>5</b>
1.1. GENERAL .....	5
1.2. DEPENDABILITY PROPERTIES.....	5
1.3. LIST OF RELATED DOCUMENTS .....	6
1.4. ACRONYMS AND ABBREVIATIONS.....	6
<b>2. P2P ARCHITECTURES .....</b>	<b>7</b>
2.1. CHARACTERISTICS OF P2P ARCHITECTURES .....	7
2.2. DECENTRALISED ARCHITECTURES .....	9
2.2.1 <i>Direct Communication (a)</i> .....	9
2.2.2 <i>Structured Indirect Communication (b)</i> .....	10
2.2.3 <i>Unstructured Indirect Communication (c)</i> .....	10
2.3. SEMI-CENTRALISED ARCHITECTURES .....	11
2.3.1 <i>Single Centralised Index Server (d)</i> .....	11
2.3.2 <i>Computational without Autonomy (e)</i> .....	12
2.3.3 <i>Computational with Autonomy (f)</i> .....	13
2.3.4 <i>Multiple Server Nodes (g)</i> .....	13
2.4. DISCUSSION OF P2P ARCHITECTURES .....	14
<b>3. PROPERTIES OF P2P SYSTEMS .....</b>	<b>15</b>
3.1. EXTERNAL PROPERTIES .....	15
3.1.1 <i>Reliability</i> .....	15
3.1.2 <i>Scalability</i> .....	15
3.1.3 <i>Security</i> .....	16
3.1.4 <i>Survivability</i> .....	18
3.1.5 <i>Safety</i> .....	18
3.1.6 <i>Maintainability</i> .....	19
3.1.7 <i>Availability</i> .....	19
3.1.8 <i>Political and legal compliance</i> .....	19
3.1.9 <i>Anonymity</i> .....	20
3.1.10 <i>Manageability</i> .....	20
3.1.11 <i>Repairability</i> .....	21
3.1.12 <i>Person centric addressing</i> .....	21
3.1.13 <i>Trust</i> .....	22
3.2. INTERNAL PROPERTIES .....	23
3.2.1 <i>Network evolution</i> .....	23
3.2.2 <i>Legacy versions</i> .....	23
3.2.3 <i>Fault Tolerance</i> .....	23
3.2.4 <i>Connection bandwidth</i> .....	23
3.2.5 <i>Intermittent node connectivity</i> .....	24
3.2.6 <i>Peer Discovery</i> .....	24
3.2.7 <i>Node addressing</i> .....	25
3.2.8 <i>Load balancing</i> .....	25
3.3. HYBRID PROPERTIES .....	26
3.3.1 <i>Responsiveness</i> .....	26
3.3.2 <i>Responsibility, accountability and reputation</i> .....	26
3.3.3 <i>Data integrity</i> .....	27
3.3.4 <i>Adaptability</i> .....	27
<b>4. ARCHITECTURE AND PROPERTY ANALYSIS.....</b>	<b>28</b>
4.1. DECENTRALISED ARCHITECTURES .....	28
4.1.1 <i>Direct communication architecture (a)</i> .....	28
4.1.2 <i>Indirect communication architectures (b + c)</i> .....	36

---

4.2. SEMI-CENTRALISED ARCHITECTURES .....	44
4.2.1 Reliability .....	44
4.2.2 Scalability.....	45
4.2.3 Security.....	45
4.2.4 Survivability .....	45
4.2.5 Safety.....	45
4.2.6 Maintainability.....	46
4.2.7 Responsiveness.....	46
4.2.8 Responsibility, accountability and reputation.....	46
4.2.9 Availability.....	46
4.2.10 Fault Tolerance .....	47
4.2.11 Political and Legal Resistibility.....	47
4.2.12 Data integrity.....	47
4.2.13 Connection bandwidth.....	48
4.2.14 Intermittent node connectivity .....	48
4.2.15 Peer Discovery .....	48
4.2.16 Anonymity.....	49
4.2.17 Node addressing .....	49
4.2.18 Load balancing.....	49
4.2.19 Manageability.....	50
4.2.20 Adaptability .....	50
4.2.21 Repairability.....	50
4.3. SUMMARY OF ANALYSIS.....	51
<b>5. DISCUSSION .....</b>	<b>56</b>
5.1. TRANSACTIONS, CONCURRENCY, AND RECOVERY .....	56
5.2. SOFTWARE ARCHITECTURES .....	58
<b>6. CONCLUSIONS .....</b>	<b>60</b>
<b>7. APPENDIX A – COMMON SECURITY RISKS .....</b>	<b>63</b>
7.1. CAUSES .....	63
7.1.1 IP Spoofing.....	63
7.1.2 Packet sniffing.....	63
7.1.3 Email spoofing.....	64
7.2. HAZARDS AND VULNERABILITIES .....	64
7.2.1 Trojan horse programs.....	64
7.2.2 Remote administration and back door access.....	64
7.2.3 Email virus spreading .....	64
7.3. ACCIDENTS, EXPLOITS AND ATTACKS .....	65
7.3.1 DoS - denial of service .....	65
7.3.2 DDoS - Distributed Denial of Service .....	65
7.3.3 Email spamming and bombing.....	65
7.4. GENERAL SECURITY SOLUTION AREAS .....	65
<b>8. APPENDIX B – MISCELLANEOUS ISSUES WITH P2P ARCHITECTURES .....</b>	<b>67</b>
8.1. CIRCUMVENTING FIREWALLS AND NAT .....	67

## 1. Introduction

### 1.1. General

When designing a Peer-to-Peer (P2P) system the choice of underlying logical network architecture plays an important role. However, no single architecture is suitable for all types of possible P2P applications. For example, applications such as Napster [1] ultimately benefit most from a semi-centralised architecture, whereas FreeNet [2] is most suitable for a decentralised architecture.

This document aims to assist in deciding which architecture would be most suitable for a given application by identifying the key P2P architectures that exist and the properties that they possess. It should be noted that this document considers these architectures being used in generic contexts, and does not consider their use in specific environments or for specific applications (for example a document management tool used inside a company).

The document begins by first identifying the key architectures that can form the basis of P2P systems. It then moves on to identify the specific properties that can apply to P2P systems in general. An analysis is provided that examines these properties and the effect the different types of architectures can have on them. Finally the document moves on to discuss other important issues including concurrency and software architectures.

### 1.2. Dependability properties

A system's dependability can be thought of as being the trustworthiness of the system, i.e. a system is dependable if it is trusted by its users to deliver the services they need and meets their Quality-of-Service (QoS) requirements. The difficulty when attempting to measure the dependability of a system is that it is context sensitive. As trust is a relationship between a system and a user, one user might regard a system to be dependable for the particular activities they use it for, another user might regard it to be totally undependable for their activities.

As well as being context sensitive, dependability is also regarded as being multi-dimensional, in that it can be influenced by a variety of system properties, such as security, reliability and availability. Because of their influence it is difficult to consider dependability without also considering these additional properties. To further complicate matters, these properties in turn can be influenced by other properties. Consequently, a system's dependability can often be influenced by properties that you would not typically regard as being dependability properties at first glance. For example, a P2P system's dependability can be influenced by its responsiveness that in turn can be influenced by the node discovery mechanism used.

Because of this network of property influences within a system, this document does not just consider what are regarded as the obvious dependability properties, but attempts to identify and consider all of the properties that exist within P2P architectures, and to relate these to the dependability of the system.

### **1.3. List of related documents**

1. P2P Architect Technical Annex, submitted 17-SEP-2001
2. Peer-to-Peer: An Overview, Lancaster University, 2001
3. Dependability Properties of P2P Architectures, James Walkerdine, Lee Melville, Ian Sommerville, In proceedings of IEEE P2P2002, Sweden, 5<sup>th</sup> – 7<sup>th</sup> September, 2002

### **1.4. Acronyms and Abbreviations**

CA	Certificate authorities
DoS	Denial of Service
DDoS	Distributed Denial of Service
DNS	Domain Name System
NIC	Network Interface Card
IP	Internet Protocol
P2P	Peer-to-Peer
QoS	Quality of Service

## 2. P2P Architectures

In this section we discuss the different logical architectures a P2P system can build upon. It first highlights the fundamental physical properties of P2P systems before discussing and classifying the different logical architectures that are commonly used. We refer to these architectures as being logical architectures as they operate above the actual physical network layer.

### 2.1. Characteristics of P2P architectures

Peer-to-peer systems are built up around a collection of nodes that are networked together in some fashion. These nodes are typically personal computers but there are no reasons why they cannot be anything with a 'digital heartbeat', be it PDA's, sensors, consumer electronics, network routers or storage systems. Communication that passes between these nodes can, for example, involve the transference of data or the relaying of commands from a control node to a slave node (for example, in some computational systems).

When considering P2P architectures there are some fundamental issues that should be taken into account. These are discussed below as well as indicating, where necessary, the stance this document takes in relation to them.

- *Logical and Physical architectures*

It is possible to consider the architectures used within P2P systems in terms of the actual physical network connections, or in terms of the logical connections. A physical architecture would represent the nodes and the physical links between them, i.e. how the data is routed between nodes, etc. A logical architecture can represent an abstraction of the physical architecture that considers just the nodes and the connections between them. For example, although a P2P architecture might only contain five nodes, the actual networking of packets between these nodes could involve machines that do not fall into this architecture.

The document focuses only on logical P2P architectures due to the fact that the physical architectures will essentially be rigid, and largely outside of the P2P system developers control.

- *Building functionality into the architecture*

Logical architectures raise the possibility of incorporating additional functionality within them. Such functionality could include the intelligent routing of messages, or some form of implicit node searching. The problem with adding such functionality to the logical architecture is that it can be potentially limitless, resulting in either bulky logical architectures or the creation of an entirely new layer above the architecture. Because of this, this document only considers stripped down logical architectures with little or no additional functionality built in.

- *Dynamic and Static P2P architectures*

Although P2P architectures are typically dynamic in nature (i.e. nodes can be easily added/removed, connections made/broken), there is no reason why a P2P system can not be largely static. For example, a system that is used within a company. When discussing the different architectures, this document considers their adaptability and their suitability for dynamic and static systems

- *Actual and Potential participation within a system*

The number of participants (be it the users or just the nodes themselves) within a P2P system can be considered from two perspectives. The *actual* number that are currently participating within the system, and the *potential* number that could participate within the system. The potential number represents the maximum number of participants who could in theory make use of the system at one time (although this might be very unlikely to actually happen). For example, a P2P system might be installed on 50 machines, but only 20 users are using it at the moment. In this case the actual participation is 20, but the potential participants is 50.

When designing a P2P system it would be wise take into account the potential number of participants within the system. Obviously for software that is to be freely downloadable over the internet this can be difficult.

- *Mapping between users and peer nodes*

When considering P2P systems it does not necessarily have to be the case that only one user has access to a node at a time. It could be the case that a node has no users (for example, if it is being used for purely computational activities), or a node could support a number of users (for example, a node that offers a service to users).

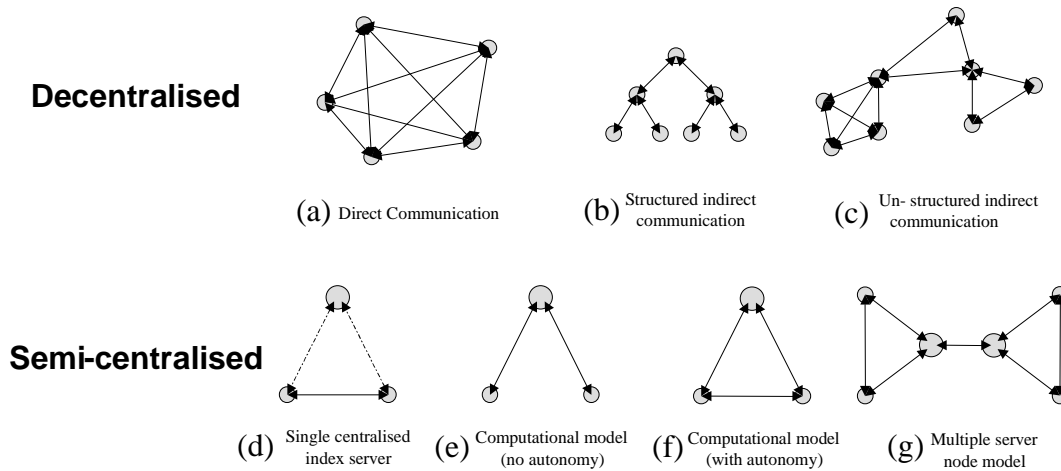
- *The notion of equality*

In a true P2P system all nodes should be regarded as being equal within the network. That is, all nodes should possess equal status, all nodes should participate equally within the network, and no one node should possess more authority than the rest. In practice, however, achieving this equality is very difficult. Although it may be possible to ensure equal status, that fact is nodes are likely to have different resources at their disposal (bandwidth, processing power, etc). This can make it difficult for nodes to participate equally within a system. For example, in a computational system it is likely that those nodes with more processing power will do more computation than those with less.

From examining existing peer-to-peer systems it is apparent that two core types of logical architecture exist. *Decentralised* (also referred to as *Pure P2P*), where each node within the network is regarded as an equal and no control nodes exist, and *semi-centralised*, where there exists at least one control node that performs an authoritative role within the network. The latter type of architecture is also referred to as a *Hybrid* architecture (a hybrid of decentralised and centralised architectures), however this is a very broad definition and can encapsulate architectures that have quite different properties. Furthermore it does not take

into account P2P systems that really *do* use a hybrid of decentralised and semi-centralised architectures.

Figure 2.1 illustrates seven possible peer-to-peer network architectures and their key characteristics are detailed below.



**Figure 2.1 – P2P Architectures**

## 2.2. Decentralised Architectures

A decentralised peer-to-peer architecture is one that does not contain any central point of control or focus. Each node within the network is regarded as being of equal standing. Figure 1 contains three decentralised network models (a to c). Each of these is broken down below.

### 2.2.1 Direct Communication (a)

In this type of architecture all nodes are regarded as equal and autonomous (i.e. independent, intelligent, etc). There is no single node that maintains any control over the network, and any data and computation that is required is spread over all the nodes.

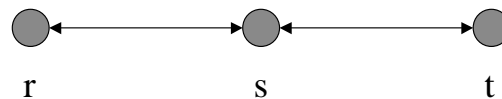
The architecture is referred to as *direct communication* due to the fact that nodes can communicate directly with each other, and hence are aware of each other. The main disadvantage of this architecture is that of scalability. Although acceptable in small-scale environments, when the architecture is scaled up to include thousands or hundreds of thousands of nodes, then it becomes unfeasible for each node to ‘know’ every other node. For example, should a node become disconnected from the network, then every other node on the network needs to be informed about it. The larger the network, the greater the overhead, and the more impractical this becomes. It is feasible, however, for this architecture to be used for systems in environments such as small companies where relatively few nodes are liable to exist.

Currently there are no known commercial systems that adopt this architecture.

### 2.2.2 Structured Indirect Communication (b)

As with the direct communication architecture described above, in this architecture all nodes are regarded as equal and autonomous. There is no single node that maintains any control over the network, and any data and computation that is required can be spread out over all the nodes.

What differentiates this architecture is that it is not necessarily the case that all nodes can communicate directly with one another. Instead, nodes that are not directly connected to one another can communicate by sending messages via another node. So, for example, in the diagram below node *r* is not directly connected to node *t* but it can still communicate with this node by having its messages relayed by node *s*.



**Figure 2.2** – Indirect communication

Various types of network topologies could be adopted, including hierarchy, ring and star structures.

Using an indirect network architecture can overcome the scalability problem that hinders a direct communication approach. However, indirect communication architectures are particularly adaptable and have been known to *evolve* over time (see section 3.2.2). Consequently a degree of management may be required to ensure that the architecture's structure persists.

An example system that adopts a structured indirect communication architecture is the Domain Name System (DNS) [3]. DNS is a system that blends peer-to-peer networking with a hierarchical communication model. When an Internet host wants to know the address of a given domain, it queries its nearest name server for the address. If that server does not know it then it is passed back up to a higher name server within the hierarchy. What is particularly notable about the DNS is that it still functions despite having scaled to 10,000 times its original size.

### 2.2.3 Unstructured Indirect Communication (c)

As with both previously described architectures, in this architecture all nodes are regarded as equal and autonomous. There is no single node that maintains any control over the network, and any data and computation that is required is spread over all the nodes.

As with the structured indirect architecture, it is not necessary the case that all nodes can communicate directly with each another. Likewise, nodes that are not directly connected to one another can communicate by sending messages via another node. What makes this architecture different is that the architecture is not structured in any way and so parts of the architecture could possess a high density of nodes, and others less so.

This unstructured nature of the architecture removes the need for any form management to enforce a network configuration, however it does result in more emphasis being placed on a discovery service given that changes to the network can happen without any other nodes being aware of it.

An example system that adopts an unstructured indirect communication architecture is FreeNet, a system used for distributing files. It is built using an unstructured indirect communication network, where each node maintains a local cache of other nodes that they know. If a node wishes to search for a document, a search request is first sent to all the nodes it knows, who in turn send it to all the nodes they know. Hence the search request is propagated through the network until the document is found or the search times out.

### 2.3. Semi-Centralised Architectures

A semi-centralised peer-to-peer architecture is one that contains at least one central point of control or focus. The purpose of these control nodes can range from maintaining a stricter control over the whole network, to simply acting as a central reference point for the remaining nodes.

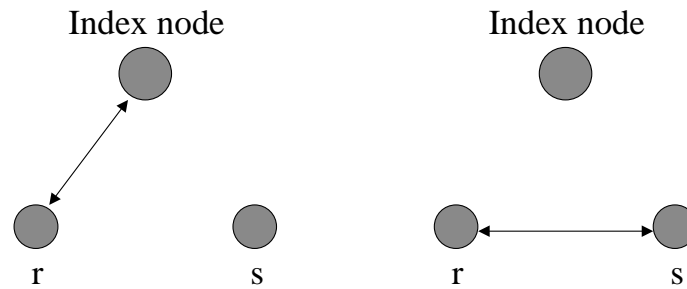
Figure 2.1 contains four semi-centralised network models (*d* to *e*). Each of these is broken down below.

#### 2.3.1 Single Centralised Index Server (*d*)

In this type of architecture there exists a single node that can act as a lookup for all the other nodes within the network. For example, this node can maintain an index or catalogue and typically behaves as a point of reference for the data or processing that is available on the network.

Only one node within the network has this capability. The other nodes are regarded as being equal and autonomous, and any data or computation that is required can be spread out over all these nodes.

All nodes within the network can communicate directly with each other, however this is typically achieved with the aid of the central index node. A typical scenario would be where one node requests the location of the node they wish to communicate with, from the central index node. Once a response is given, the node can then establish a direct connection with the target node. This is illustrated in figure 2.3, where in the left illustration the node *r* is requesting the location of the node *s* from the index node. Once this has been obtained *r* can set up a direct connection with *s* (right illustration).



**Figure 2.3** – Using the Index node to establish direct communication between peers

By using a central index node the need for a discovery service is reduced (or even removed). When a node connects to the network it informs the index node of its location. This means that establishing a communication link between two nodes can be a simple and quick process. However the central index node does pose as a potential single point of failure. Should it fail then the remaining nodes of the network will lose their means for looking up other nodes on the network. To take this into account, a combination of a lookup index node and a discovery service can be used.

An example system that utilises such an architecture is Napster. Napster is a music file sharing system that uses a central index node to maintain a database that contains details of what music files each node within the network holds. A user can then interrogate the central index node to locate the nodes that possess the music they are looking for.

### 2.3.2 Computational without Autonomy (e)

The two computational architectures are similar to the single centralised index server discussed above. As with this architecture, a single node within the network acts as a focal point for all the other nodes.

However for this particular architecture the remaining nodes of the network do not possess their own autonomy and are essentially controlled by the central node. For example, this central node can control and manage the distribution of data and assign processing tasks to each node within the network. Furthermore communication between nodes must be through the central index node.

Arguably, this type of architecture should not be considered a proper P2P architecture given that the nodes within the network do not possess their own autonomy and essentially act as slaves to the central node. However, applications have already been developed that adopt this architecture and these have been labelled as being P2P applications. Consequently this type of architecture has been included.

An example system that utilises such an architecture is SETI@home [4]. SETI@home is a system that makes use of a node's processing power when it is not being used (in this case when the screen saver on the computer has been activated). A central server node distributes the data to be processed to the nodes within the network, and then collates the results

together. The user on each node has no control or knowledge of what processing is being done.

### **2.3.3 Computational with Autonomy (f)**

The computational with autonomy architecture is identical to the architecture described in 2.3.2, except that all nodes retain a degree of autonomy. This autonomy could allow the node to operate intelligently and actually make requests to the server rather than simply being dictated to, or allow the node to communicate with other nodes on the network. Again in this circumstance it is likely that any communication between nodes will be facilitated by the central node (as described for the single centralised index server architecture).

This architecture has the advantage that the nodes within the network are not forced to be merely slaves to the central node. Essentially it is the same as the single centralised index architecture except it is geared more towards distributed computation.

Currently there are no known commercial systems that adopt this architecture.

### **2.3.4 Multiple Server Nodes (g)**

All of the semi-centralised architectures that have been discussed up to now have been based on the assumption that there exists only one central node. There is no reason, however, why more of these central nodes cannot exist within the network. This has advantages such as potentially increasing a P2P systems reliability by removing a potential single point of failure, or if the nodes happen to be physically quite spread out having multiple central nodes could help tackle any loss in Quality-of-Service.

Communication within such an architecture can be similar to the single centralised index server architecture, with the difference that the central nodes can also be connected together. This can mean that the control nodes can essentially pool their knowledge when it comes to issues such as node locations.

An issue with this architecture is of how the nodes connect to the server nodes. If they are only connected to one server node then this can become a potential failure point, however allowing all nodes to connect to all servers might limit the advantages this architecture can provide.

This type of architecture can also allow for a hybrid of other types of architectures. For example, the server nodes could make use of a direct communication architecture amongst themselves, but collectively act as a single server node within a semi-centralised architecture.

Currently there are no known commercial systems that adopt this architecture.

## **2.4. Discussion of P2P Architectures**

Not all of the architectures discussed above would be suitable for every type of peer-to-peer application. Model A's requirement of allowing every node direct communication with every other node in the network, is not particularly scalable. Out of the three decentralised models, B and C are the more practical and the development of existing applications based on these models (for example Freenet, DNS) supports this.

The biggest factor in application suitability is whether the architecture used is based on a decentralised model or a semi-centralised model. Both have their advantages and disadvantages and consequently better suit different types of application. The following sections identify the key properties of P2P architectures and assesses how they can be affected by the different architectures.

## 3. Properties of P2P Systems

This section highlights the properties that may apply to P2P systems. It splits the properties into the following categories.

- *External properties* – system properties that can only be viewed externally (for example, by the user)
- *Internal properties* – system properties that can only be viewed from within the system (for example, by a system component)
- *Hybrid properties* – system properties that can be viewed both internally and externally

A short description is provided for each property.

### 3.1. External properties

#### 3.1.1 Reliability

Reliability is arguably the most important characteristic of almost all software systems. As users expectations of quality have increased, it is no longer acceptable to deliver software that can fail on a regular basis. Unreliable software can result in high costs for the end-users, and can give the software developers a bad reputation for quality. However, it is difficult to define reliability precisely. More formally, a reliable system should conform to its specification, however essentially, you can think of it as the extent to which a system behaves as expected by a user [5].

The reliability of a software system is not a simple measurement that can be associated with the system. A system's perceived reliability is often based on its pattern of usage. It is quite possible for a system to manifest faults yet be seen as reliable by users of the system.

Reliability is important within a P2P system due to its distributed nature. A P2P system is particularly prone to network failures and so mechanisms need to be considered that can reduce the impact on the system should one occur.

#### 3.1.2 Scalability

Scalability is the ability of a system to operate without a noticeable drop in performance despite increases or decreases in its overall operational size. For example, this could be whether or not a system that was originally intended to be used by 10 people can still operate satisfactory when a thousand people are using it.

Peer-to-peer systems, by their very nature, are designed to be distributed over many nodes. Existing P2P applications have shown to what scale a P2P application can potentially grow to (Napster has had over 40 million client downloads, SETI@home has had over 2.5 million, though this does not represent the number of clients in use at one time [6]).

Scalability within P2P systems can take on numerous aspects: scalability in respect to the number of users using the system, scalability in respect to the number of nodes within the system. In addition there is also catering for the *potential* scalability of the system with respect to these two aspects (as mentioned previously).

Accordingly, catering for scalability should play a significant role when designing a peer-to-peer system.

### 3.1.3 Security

The level of security within a system represents its ability to protect itself against accidental or deliberate intrusion [7]. It is possible to break security down into three subclasses, integrity, confidentiality and privacy. Integrity is the system's ability to protect itself from damage (data integrity is an example of this, discussed in section 3.3.3). Confidentiality is the system's ability to ensure that information is only revealed to those people who have a right and a need to access that information. Privacy is the art of keeping information secluded from the presence or view of others, this information may be personal to an individual or organisation, and could include the anonymity of users within a P2P system (discussed in section 3.1.9). Secrecy is the act of keeping information in a state of concealment, once achieved access should only be allowed through proper access methods known as authorisation.

Ensuring security within P2P systems is more difficult than with standard centralised server systems. On the one hand there are issues such as authenticating your communication partners and sharing information only with the people you trust. And on the other the nature of P2P systems typically involves supporting the anonymity of users.

Security requirements can differ from one system to another, based upon what assets require protection. Some of these assets can even contradict one another (i.e. user anonymity and user authentication). The first goal in building a secure system is identifying the assets that should be protected.

Vulnerabilities are ways in which potential attackers can penetrate a system. Vulnerabilities can occur in a system through many ways: design errors, implementation errors, change in the organisational/computing environment in which the system exists and administration errors, or a combination of them. These are largely generic to any system, including P2P systems. An example of vulnerability was detected in ICQ where a buffer overflow could allow an attacker to execute arbitrary code on a victims system [12].

Attacks resulting from vulnerabilities in a system are typically performed for some hostile goal. These attacks can originate from within the system/organisational boundaries (e.g. employee saboteurs) as well as from outside these boundaries. Single stage attacks are

usually where a vulnerability is known and is immediately exploited, and there are two (or multi) stage attacks where the first attack is simply intended to discover vulnerabilities in the system that could be exploited. Two (multi) stage attacks are commonly used on systems where little is known about their vulnerabilities.

The goals of attacks can be broadly placed into areas such as 'Denial of Service', 'data leakage' and 'data corruption'. These goals are the consequences of the attacks and are the main objective of the attacker.

The links between vulnerabilities, attacks and consequences are not clear-cut however. One point is that vulnerabilities may exist that are never known, found or exploited, which always leaves the potential for vulnerabilities and is why any security practices should not be stopped once a system is in place.

There is also no need for a vulnerability to exist for an attack to take place. Professionals, hired by an organisation, can carry out attacks on a system to discover vulnerabilities that can then be patched. Also the consequences of an attack will clearly not transpire if the attack is successfully repelled by the system, or more dangerous, is the fact that the consequences will not be detected until it is too late. This clearly shows that although relationships exist between vulnerabilities, attacks and consequences these relationships are not carved in stone.

Because a P2P system can be viewed as a synergistic system, the emergent system becomes composed of the individual nodes that are connected to it. This can create security problems, as it does in any component-based system incorporating message-passing communication, where the messages that are passing around are exposed to risks of tampering, sniffing and falsification. Security considerations are often incorporated into specific areas of a system (the parts containing the assets to be protected) and this philosophy can sometimes be inappropriate, especially in a P2P environment. The entire system must be taken into account, and the systems architecture from its foundations must be geared to support the security design.

Because P2P is essentially a new technology it raises more concerns, as the application of traditional security techniques to this new technology may not be effective against new and innovative methods of attack. Traditionally security in a system becomes more sturdy as time passes, new vulnerabilities are patched and successful attacks become harder to achieve, P2P is so young it has not had time to mature its basis of security. There are also several perspectives to security, that of the attacker, the user and also those responsible for the implementing the security (both software developers and administrators). These perspectives may or may not have an impact on each other depending on the system; a user for example may want a certain level of security out of the system but without it hampering their usage of it. Likewise a system administrator may want to implement a high level of security within a system, but this may come at the expense of usability. An important point is the fact that administrators creating usable systems, at the expense of security, make the attackers job easier.

Typical security risks and possible solutions for are discussed in more detail in the appendix of this document.

### 3.1.4 Survivability

Survivability is the capability of a system to fulfil its mission in a timely manner in the presence of attacks, failures, or accidents [8]. The mission of a system is defined to be a set of very high-level requirements or goals. The notion of survivability itself draws upon other properties including security, fault tolerance and reliability, thus making it heavily dependent on how these other properties are tackled within the architecture.

To maintain their ability to deliver their services, for a system to be survivable it must exhibit four properties [8]:

- *Resistance to attacks.* The system must possess strategies for repelling attacks. This draws upon the security property.
- *Recognition of attacks and the extent of damage.* The system must possess strategies for detecting attacks and to evaluate the extent of any damage. This draws upon security and fault tolerance properties.
- *Recovery of full and essential services after an attack.* The system must possess strategies for restoring compromised information or functionality, whilst limiting the extent of the damage. This draws upon the fault tolerance and reparability properties.
- *Adaptation and evolution to reduce effectiveness of future attacks.* The system must possess strategies for improving survivability based on experienced gained from attacks.

Survivability in P2P systems raises some interesting issues, as in some cases the inherent redundancy can help attain survivability, whilst the lack of a central control can hinder it. The choice of architecture can affect how easy it is to achieve these four properties.

### 3.1.5 Safety

The level of safety within a system represents its ability to operate without catastrophic failure [5]. For example, a system that controls an airplane's flight systems is regarded as being a safety-critical system due to the fact that the software can pose a threat to human life. Safety-critical software systems fall into two classes, primary safety-critical and secondary safety-critical [5]. The primary class represents systems that would result in human injury or environmental damage should they malfunction. The secondary class represents systems that can indirectly result in injury due to a malfunction. For example, a malfunction within a system that maintains a medical database could result in the wrong dosage being administered.

Although currently developed P2P applications have not been safety-critical, it is unwise to assume that this will always be the case. Consequently safety could have an important influence on the P2P architecture used.

### **3.1.6 Maintainability**

Maintainability represents the ease in which the system can be changed after it has been delivered and is in use [5]. Such changes could involve the simple correction of coding errors, through to the correction of design errors, the correction of specification errors, or the introduction of new system requirements.

Three different types of software maintenance exist, corrective, adaptive and perfective [5]. Corrective maintenance deals with the fixing of errors within the software. The higher level the correction the more expensive it tends to be to fix. Adaptive maintenance deals with altering the software so that it can operate in a new environment, for example on a different hardware platform. Perfective maintenance deals with the implementation of new functional or non-functional requirements. These are normally generated when the nature of the end-users organisation of business changes.

One major issue with maintaining a P2P system is updating the software on all the nodes of the network. This could be an automatic update, or a manual one that is performed by the user. For the latter it cannot be assumed that all users would perform the upgrade and so the system may have to be able to cope with different versions of itself. Furthermore, the lack of updating the system could be particularly important should the update patch a security flaw within the system.

### **3.1.7 Availability**

Availability is the probability that a system, at a point in time, will be operational and able to deliver the requested services [5].

The availability requirement of a P2P system, like any other system, is dependent upon the use for which the system is deployed. Traditionally, high availability requirements can usually be catered for by using replication or by employing fast methods of re-initialising the services that become unavailable.

Replication can, theoretically, be utilised by a P2P system because the systems very nature can actually support it, in fact replication is almost a by-product of such a system and as such could be exploited for availability purposes. Re-initialising a P2P system on the other hand could be more of a problem, especially for distributed applications where only some of the nodes are re-initialised (this also relates to transactions – see section 5.1).

### **3.1.8 Political and legal compliance**

In some circumstances it might be the case that political or legal actions need to have an effect on a system. For example, a court order might be given for the system to be shut down, or its data be made available for examination.

P2P systems can be more resilient to such actions due to their distributed nature. It would be impossible to shut down every node on a P2P network that spanned hundreds of thousands of

nodes. Similarly it would be impractical to attempt to obtain the data that is held on every node.

The type of architecture used can affect a P2P systems compliance. Shutting down a truly decentralised system would be difficult (depending on its size), whereas a semi-centralised system can usually be significantly hampered by the removal of the central index node.

It might also be necessary to monitor a P2P system (by say a third party) to ensure it is complying politically and legally. Again this can also benefit from the type of architecture used, with it likely to be easier to achieve in semi-centralised systems.

### **3.1.9 Anonymity**

Peer-to-peer applications may well need the ability to hide a user's identity without compromising authentication. In addition it may also be desired for data held within a peer-to-peer system to be kept in an anonymous state.

For hiding user's identity, pseudonyms rather than IP addresses can be used to identify users. Furthermore, mapping data communications through disinterested and trusted third party proxies can also help.

Data can be hidden (using encryption technologies) from the node hosting it. This would disallow the node from knowing the contents of the hosted data. For example, data can be held on nodes using a standard filename (such as '*data*'), there should be no file extension that would allow the recognition of the type of file being hosted. Retrieval could still be efficiently performed by storing the files in directories that are named as the hash value of the file being contained.

In terms of network architecture, a semi-centralised model could arguably provide the most support for anonymity. Users can register their details with the index node, but ask that these be not made available to any other users. Yahoo uses such an approach. The advantage of the semi-centralised approach is that there exists one point where the details of each node exist, this can be important for certification and reputation tracking. With a decentralised architecture this information can be withheld entirely from the rest of the network.

### **3.1.10 Manageability**

Manageability reflects the ease in which the system as a whole can be managed. The management of a system could include controlling the flow of data and assigning access rights, through to managing the maintenance of the system. This, for example, can be particularly important in business environments where managers might need to ensure rigid control over the system.

Given that nodes within a P2P system possess a degree of autonomy it can be difficult to provide the same level of manageability that more traditional client-server systems might possess. Without a single centralised location for the storage of data it can be hard to track

which nodes possess a copy of specific data and how up to date it is. Similarly it cannot be assumed that all nodes within a P2P system will regularly update the software that they are running (see 3.1.6).

### 3.1.11 Repairability

A system is regarded as being repairable if it allows defect correction with limited effort. Repairability, itself, can be further broken down into a further set of properties:

- *Diagnostic ability* – identifying the defect
- *Accessibility* – how easy it is to access the defected part of the system.
- *New component availability* – whether replacement components for repairing the defect exist or are easy to obtain
- *Re-start ability* – how easy it is to restart the system once a repair is made

How a system is designed and implemented can greatly enhance the repairability of a system. In particular, modular and abstract designs can assist repairs by allowing fault-prone components to be easily replaced in the event of a failure.

Repairability draws on both fault tolerance and maintainability properties. A repairable system needs to be able to detect faults within the system and then perform corrective maintenance in order to repair them. Repairability and system reliability are also strongly linked, with the need for repairability decreasing as a system's reliability increases.

As P2P systems are distributed in nature the main difficulty when considering repairability is how to identify and repair defects that can occur throughout the network. This is especially the case for decentralised architectures.

### 3.1.12 Person centric addressing

Traditionally, individual machines (nodes) have possessed an IP address that is used to uniquely identify them on the network. However, the designers of the existing IP infrastructure had not predicted nor taken into account the vast uptake of the Internet and the consequent number of IP addresses that would be required. Because of factors such as, the increase in modem users, the old practice of providing every host with a fixed IP address is no longer feasible for the simple reason that not enough IP addresses exist.

This problem has largely been circumvented with the use of dynamic IP's (a hosts address that frequently changes), but dynamic IP's pose their own problems for peer-to-peer applications as they can make machines difficult to reach (discussed more in section 3.2.7). Another technical solution is with the introduction of IPv6, the next generation Internet protocol. IPv6 uses 128-bit addresses that will provide enough permanent IP addresses for

every host on the Internet. The main disadvantage of IPv6 is the complexity of the changeover and as a result it is still unknown whether it will be commonly deployed.

Existing peer-to-peer applications have had to tackle the addressing issue themselves and have focused on implementing a person centric addressing layer on top of the machine centric addressing layer. Rather than assigning a unique ID address to each machine, an addressable ID is assigned to each user. This is particularly important as it becoming increasingly common for a user to use more than one machine (e.g. a desktop machine and a laptop machine). A user might be using one IP address in the morning and a totally different one in the afternoon.

This is analogous to the change brought by the introduction of mobile phones. Prior to mobile phones a phone number was attached to particular physical location. With the advent of mobile phones a phone number can now be dynamically mapped to the location of the phone's owner. Existing peer-to-peer applications act in a similar fashion. An address is mapped to the user, not to the machine, no matter where the user is. [6]

### **3.1.13 Trust**

Trust is a property that can depend on a range of other properties. For example, trust can be influenced by the perceived dependability of the system, its security measures, or the behaviour of other users of the system. Actually defining and measuring trust is a difficult task due to its highly subjective nature. For example, a pilot might 'trust' his co-pilot in every day life, but when preparing to fly a plane he has to intentionally not trust him and double check all his pre-flight checks. Furthermore, attempting to quantitatively measure this level of trust is also practically (if not totally) impossible. Because of this no attempt is made to define it here.

In traditional computing trust is implemented through certification authorities, entities that can vouch for a certain user/device. Certificate authorities (CA) are linked together in a hierarchal fashion to form 'chains of trust' to allow users/devices to authenticate and administer a domain of trust between themselves and others within that chain.

The structures of these 'chains of trust' are built upon the current commercial Internet infrastructure. P2P systems are essentially ad hoc adaptive networks, and using centralised certificate authorities for authentication purposes is quite obviously a major problem, as it would involve knowledge of the network infrastructure in advance.

To allow the inclusion of CA's within a P2P infrastructure would likely involve the leverage of some aspect of the network architecture and/or application domain.

## **3.2. Internal properties**

### **3.2.1 Network evolution**

Studies of existing P2P systems have shown that the architecture used can evolve over time (Gnutella [9], and chapter 8 of [6]). Although P2P systems set out to give all their nodes equal status, in reality this will never be totally achievable due to the different resources available to each node. These resources can range from network bandwidth, hard disc space, to processor power.

From studies of Gnutella it was observed that the nodes with more resources would move to the centre of the logical network and those with the least would move to the edge. This is understandable as the nodes with greater bandwidth are much more likely to act as routers to other nodes.

### **3.2.2 Legacy versions**

It is likely that as new versions of the P2P software are released, not all nodes within the network will upgrade. Consequently you could have the scenario where multiple versions of the software are run across the network. A P2P system needs to be able to still operate despite the different versions of the software that might be running on the nodes. An example of an existing P2P system in which this can be observed is ICQ [10] where users can still communicate with each other despite running different versions of the software on their machines.

### **3.2.3 Fault Tolerance**

Fault tolerance is the ability for a system to continue giving a correct service following the manifestation of a fault or faults either through errors in the system design, implementation or introduced following an attack [5]. Fault tolerance is important in situations where a system failure could cause a catastrophic accident or large economic losses. For example, the computer systems within an air traffic control system must be continuously available.

Fault tolerance can be split into four aspects. Failure detection, the systems ability to detect that a failure has or is about to occur. Damage assessment, identifying the parts of the system that have been affected by the failure. Fault recovery, restoring the systems state to a known 'safe' state. Fault repair, modifying the system so the fault cannot occur again (see also reparability – section 3.1.11).

### **3.2.4 Connection bandwidth**

How the nodes are connected together in a P2P network can vary considerably, from a user connecting via a modem from home, to a machine connected via a T3 connection at a

university. Consequently the amount of network bandwidth available to a single node can vary considerably.

Ideally a P2P system should be able to operate no matter the connection bandwidth however the choice of network architecture can greatly affect a systems performance. For a decentralised architecture, if two portions of a P2P network are only connected together by a low bandwidth connection, then this could act as a severe bottleneck for peer communication.

### **3.2.5 Intermittent node connectivity**

Due to the very nature of peer-to-peer, it cannot be assumed that nodes within such a network are connected at all times. For example, a computer connected to the Napster network might be switched off when it is not in use. Consequently the MP3's that it was making public are no longer available on the network (unless provided by another node). When the computer is switched back on and the network connection re-established, the MP3's once again become available. Similarly with a computational based application it cannot be assumed that a node can always work on a computation, and thus this needs to be taken into account for distributed computations.

As with the connection bandwidth, the intermittent connectivity of a node can play an important role in what network architecture is used and can have a significant effect on a systems perceived dependability. When designing a peer-to-peer system, it is important to cater for a node's intermittent connectivity and not assume that the node will always be connected.

### **3.2.6 Peer Discovery**

One important property for a P2P system is its ability to discover the other nodes that reside on the network. Systems that are based on semi-centralised architectures can rely to a large extent on the central index node to provide the addresses of other nodes on the network. With decentralised architectures the task of discovering the location of other nodes on the network is typically spread across all nodes, as they each can maintain their own local cache of node addresses and discovery messages are sent from one node to the next until the required node is found.

The disadvantage of peer discovery using a central index is that should the index node fail then the nodes within the network can suddenly become isolated, however generally this method of peer discovery is more efficient than the decentralised alternative. The disadvantage of decentralised peer discovery is that it can be very slow, cached node addresses can quickly become out of date, and if a partition has occurred within the network then it might not be possible to locate all nodes.

### **3.2.7 Node addressing**

The rapid uptake of the Internet has resulted in a far larger demand for machine IP addresses than originally catered for in the existing network protocol design. It is no longer possible for every machine that is connected to the Internet to possess its own fixed IP address. Accordingly it is now common for many networked machines to be provided with dynamic IP addresses – an address that frequently changes.

Node addressing is a particular concern within P2P systems. The use of dynamic IP addresses can cause problems as they can make machines difficult to reach. A node on a peer-to-peer network might possess a particular IP address one day, but another the next. Consequently, unless other nodes are informed of the new address, that node can become lost from the network.

A common strategy has been to maintain a database of the current IP addresses of the nodes within the network. This is usually used in combination with a semi-centralised architecture, where the central index node maintains the database. With a decentralised architecture this becomes more difficult as the database would have to be distributed over all the nodes of the network and maintaining consistency would become an issue.

### **3.2.8 Load balancing**

Load balancing is where the load that is placed on components within a system is balanced to ensure that a component is not overworked or, alternatively, underused. For example, if a network system makes use of a group of servers to handle requests, the load each server receives is evenly distributed (as is possible) over each of the servers. In this way the scenario does not arise where one server might go down due to an overloading of requests, whilst another server is being underused.

With P2P systems load balancing can be important when deciding how much of a nodes' resources to draw upon. So for example, a node that possesses a lot of network bandwidth and processing power can take more load than a node that only has a modem connection and a slow processor. Similarly for semi-centralised architectures load balancing can come into play if there exists more than one central server node.

### **3.3. Hybrid Properties**

#### **3.3.1 Responsiveness**

Responsiveness not only includes latency, jitter and other system performance attributes but also how the end user perceives this performance and to what use the system is being employed (i.e. real time constraints). Furthermore, the consistency of this performance is also important. A system that fails in these departments can not only irritate the end users but can also have a dramatic effect on its commercial uptake.

Some architectures may be better attributed to certain types of application than others and mirroring this is the affect applications have on the network infrastructure they are built upon.

Generally users of a system will require a fast and consistent response to interactions with their local applications.

#### **3.3.2 Responsibility, accountability and reputation**

A key challenge within P2P systems is enforcing rules of social responsibility (responsibility also relates to trust, with respect to the behaviour of other users of the system. See section 3.1.13). Examples of where this can break down is where a user takes advantage of the system to send out Spam, overuse resources, or even initiate an actual attack on the system. To avoid the danger of P2P systems being overrun by such activities and to provide means to track them to the originator should they occur, the system must make use of mechanisms that can make users accountable for their actions.

In terms of providing mechanisms to achieve accountability, two main approaches exist: reputation tracking and micropayments [6] (token-based systems are another possible alternative). Reputation tracking is where a profile of each user is built up, and is held and maintained by a server node. A users reputation profile is based on third party feedback. An example reputation system is that used by Ebay, where users can leave positive or negative feedback about a user after an auction.

A micropayment is where in order for a user to make use of a resource on the network they have to exchange it with something of equal value. For example, if a user wants a server to store some data then he/she has to pay a micropayment in order to protect the server's resources from overuse. In this way resources will not be overused, nor will malicious users be able to mass distribute unsolicited messages through the network (not unless they can pay for it).

Both these strategies rely to a certain extent on the type of architecture in use. The reputation tracking mechanisms, for example, require a central location in which the profiles are maintained. Consequently the type of architecture used for a P2P system can have an effect on the type of accountability mechanisms used.

### 3.3.3 Data integrity

It is important for the data that is stored and manipulated by a system to maintain its integrity, that is the data should not become corrupt, nor become invalid due to issues such as concurrency or network problems. Should such an integrity loss occur then it could result in systems failing or in business decisions being based on incorrect data.

Maintaining data integrity is more difficult with P2P systems due to the fact that data will frequently be passed between nodes. Consequently there is the possibility that data can become corrupted in transfer or whilst residing on a node. Furthermore, as there is nothing stopping multiple instances of the data existing then there is also the issue of concurrency, where it can become unclear which node possesses the current correct version of the data.

However the distributed nature of P2P can also contribute to achieving data integrity. By having multiple copies of the data existing through out the system there is in a sense an in built backup mechanism. Should the data on one node become corrupt then it might be possible to obtain the same piece of information from another node.

Achieving data integrity is likely to be most difficult to achieve in fully decentralised systems, due to the lack of central foci. For semi-centralised systems it is different, as with such architectures there exists the possibility of the central server nodes being used to store the most up to date versions of the data. By allowing these server nodes to make backups achieving data integrity can be further supported.

### 3.3.4 Adaptability

P2P networks are very unpredictable in nature; machines with different specifications and networking capabilities, can be connected to and from them at will. A P2P system needs to be able to adapt when such changes occur to ensure its continued operation (for example, a computational system needs to adapt how it distributes processing when a new node is added to the system). Adaptability is a systems ability to adapt to a dynamically changing environment.

However adaptability is not merely restricted to changes to the network topology. It can also involve the network evolving to increase its overall efficiency. This can be important should it become apparent that certain nodes act as bottlenecks, while others are being underused. For example, if it becomes apparent that routing is being done via a node with a low bandwidth connection. Such adaptability has been shown to happen in some existing P2P systems such as Gnutella (chapter 8 in [6]). Network evolution is discussed in section 3.2.1.

How successful a P2P network is at adapting itself can be influenced by the type of underlying architecture used.

## 4. Architecture and Property Analysis

This section examines the different architectures that were identified in section 2 and discusses the effects they have on the system properties that were identified in section 3.1.

The properties, Trust, Person centric addressing, Network evolution and Legacy versions, have not been analysed due to the fact that it is difficult to assess the influence on these caused by the choice of architecture. It should also be noted that, in some cases, properties are less affected by the type of architecture used (if affected at all) and therefore, in places, the analysis is replicated. A summary of the analysis is presented in table form on page 51.

The section splits the architectures into two categories, decentralised and semi-centralised architectures, and discusses each in turn.

### 4.1. Decentralised Architectures

This section focuses on the decentralised architectures. Analysis showed significant differences between direct communication and indirect communication architectures. Consequently these have been split into separate sections.

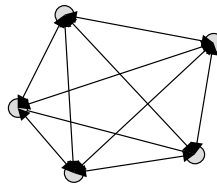
The analysis of the decentralised architectures also found that a lot of the architectural properties could be hindered due to the lack of central points within the network (i.e. a semi-centralised architecture). It could be envisioned that one or more nodes be assigned additional responsibilities within the network, for example, monitoring the network for faults, or tracking reputation. However this poses two problems:

- Makes nodes within the network unequal. The definition for decentralised architectures is that all nodes are equal. To assign one or more nodes additional responsibilities would void this.
- With the indirect communication architectures, it cannot be guaranteed that a single node would be able to communicate/monitor every other node on the network. Issues such as network partitioning could mean that clusters of nodes are un-contactable.

Consequently the analysis in this section only considers true decentralised architectures in which all nodes are equal.

#### 4.1.1 Direct communication architecture (a)

This section discusses the possible implications of the direct communication architecture on the properties that were identified in section 3. A direct communication architecture is one where all the nodes within the network can communicate directly with each other. All the nodes are regarded as being equal and autonomous, and no single node exhibits control over the network. This architecture is illustrated in Figure 4.1.



**Figure 4.1** – Direct Communication Architecture

#### **4.1.1.1 Reliability**

Reliability is a quantifiable dimension to dependability; it is a measurement of probability that the system, over a certain time period, will deliver its services as specified. A direct communication architecture needs its logical architecture to be mapped out, before it is put into operation. This means that a lot of problems encountered by other decentralised systems can be avoided, and can leave developers to worry about the other important aspects, such as reliability. In such a model, the failure of the entire system is less likely to be as problematic as other decentralised architectures, simply because it is liable to be closely administered and easier to get back up and running again. The close knit community of a direct communication architecture could help in transactional systems, where a close eye has to be kept on detecting faults that occur and their removal.

The view of reliability by a system's users depends entirely upon how and for what purpose the system is being employed. An emergency messaging system in a hospital, used for paging doctors, is more likely to be viewed with suspicion than that of a simple inter-office messaging system. The more knowledge that is known about a system, such as one built upon a direct communication architecture, can help in boosting user confidence.

#### **4.1.1.2 Scalability**

Scalability is the obvious problem in decentralised direct communication architectures. It is difficult to scale this architecture up to a large degree because of the massive strain it would put on peer node resources (i.e. keeping up-to-date lists of other nodes and the number of open port connections to other nodes). Due to this fact, this architecture is only recommended for small scale systems.

#### **4.1.1.3 Security**

The lack of any centralised control can hamper any attempt at implementing security mechanisms and certification, thus making key exchanges a problem. This particular architecture will also provide problems for anonymity due to the amount of information needed in advance for implementing the system. However the decentralised nature of the architecture can provide some advantages as there is no single point of failure that can be targeted for attack (for example, denial of service attacks).

#### **4.1.1.4 Survivability**

There are both advantages and disadvantages of achieving survivability within direct communication architectures.

Survivability itself, draws upon other architectural properties such as security and fault tolerance. These all, to varying extents, can be best achieved when used within a semi-centralised architecture due to their general requirement for central points of control. As a consequence, generally for achieving survivability within a P2P architecture it would be most beneficial to use a semi-centralised architecture.

However, decentralised architectures do not rely on any single node (or nodes) within the network. This can make them less susceptible to some forms of attack, but can also make it harder to recover from an attack should one be successful. In addition, the inherent redundancy in P2P systems also means that should a node fail or be attacked, it can still be possible for the system to function normally.

Although decentralised architectures are not the most suitable for achieving all aspects of survivability, of the three architecture types, the direct communication can possibly provide the better option. Because a significant amount of knowledge about the network topology is required at the architectures creation, it might be possible to use this knowledge as a basis for monitoring the architecture (see 4.1.1.9).

#### **4.1.1.5 Safety**

Trying to ensure safety in a system that uses this architecture could be a major undertaking, due to the fact that there are no centralised points of control. Ensuring safety typically requires these focal points for carrying out safety and system checks. Without doing these procedures it would be difficult to know when failures occur within the system. Consequently this architecture would be the least suitable as a basis for a safety critical system.

#### **4.1.1.6 Maintainability**

Maintaining a system based on this architecture can be problematic due to the lack of centralised control. However, because this particular architecture has scalability constraints it may well be small enough to be maintained manually to some degree.

#### **4.1.1.7 Responsiveness**

Because every peer's address would be known by all others in a decentralised direct communication architecture, it could be possible to leverage this to allow for the fastest possible response times for communicating peers. Node address lookups can be done locally with the use of searches implemented through a list in main memory or even disk cache, both of which are an order of magnitude faster than a network search equivalent. Furthermore

requested data that is replicated could be first sought on the same subnet as the requesting peer, this would reduce latency and more efficiently use the available bandwidth.

#### **4.1.1.8 Responsibility, accountability and reputation**

The lack of central points within the network can make it difficult to incorporate any form of reputation tracking mechanisms within the architecture. Responsibility may be achieved through some form of micro-payment facility, but to enable this would require dependence on security (i.e. encrypting micro-payment credits and their transfer mechanisms so a user cannot interfere with his/her own credit status). There is also the fact that this form of architecture is liable to be used for small scale specialised systems, and responsibility, accountability and reputation schemes may not be needed at all.

#### **4.1.1.9 Availability**

Achieving knowledge about availability within P2P architectures can involve the transmission of digital heartbeats (frequent pings) to monitors at known locations on a network; this acts as an advertisement by a node to let others know of its availability. The indirect communication architectures (b and c) lack the monitors that can be hosted on semi-centralised architectures and also the topology knowledge that this architecture (a) exhibits. Because knowledge of all other nodes on the network (their locations etc) is known when using this architecture, it would be possible to broadcast a heartbeat to each one. This will obviously incur a networking overhead, but as long as the number of nodes on the network is not too large it should be manageable.

#### **4.1.1.10 Fault Tolerance**

Achieving fault tolerance within direct communication architectures can be a difficult task due to the lack of central foci points that can monitor the status of the network. Unless a digital heartbeat (as mentioned above) is in use, a fault within the network is only going to become apparent when an attempt is made to either contact a node or use the network connection (depending on what has failed). Similarly, it is also difficult to assess the damage that has been caused by the fault.

Recovering from a fault within a direct communication P2P system can also be a difficult task as it is hard to revert back to a safe state. A possibility would be for a global alert to be sent out when a fault is discovered that can ensure that nodes revert to a safe state. However there is no guarantee that all nodes will receive this alert, and due to the lack of central control the rest of the network is not guaranteed to be informed should the fault be later fixed.

Repairing the fault is again difficult and would likely involve the person responsible for the node or network connection having to make the repair themselves. As this is not monitored, it cannot be assumed that this will be done and done properly.

This architecture could facilitate procedures such as global alerts, so at least the discovery of faults within a system could be broadcast to the other nodes. Repairing faults or performing rollbacks is in the domain of the applications themselves, negotiating repairs and informing others of these repairs are both tied to the applications and the network architecture.

The main advantage for this architecture is that it does not have the points of failure that exist within semi-centralised architectures. For these architectures a fault occurring to a server node is likely to have a far greater impact on the system, than should a fault occur within a node in a decentralised architecture. From this respect, the decentralised architectures can be more fault tolerant.

#### **4.1.1.11 Political and Legal Resistibility**

Although it is liable to be easier for any governing body to close a semi-centralised system than it is a de-centralised one, this specific architecture is effected differently due to the fact that it is unlikely to be used in a large scale environment. Consequently, although still difficult to close, it will likely to be easier with a system based on a direct communication architecture, than the other decentralised alternatives. These issues are likely to effect more commercial systems such as Napster, rather than internal business systems.

#### **4.1.1.12 Data integrity**

Data integrity within P2P systems is typically difficult to achieve and particularly in those that adopt a direct communication architecture. Data can be frequently passing between nodes and with no notion of a central repository, it can be difficult to achieve concurrency. With a direct communication architecture it is highly likely that over time data that is held by a node, despite the node owner believing otherwise, will become invalid.

However, although a direct communication architecture can suffer from concurrency issues, the fact that data redundancy is likely to occur within the system means that a form of inbuilt data backup can occur. Should the data become corrupt in one part of the system, then there is the possibility of retrieving it from another part. The problems occurring here are the same as those in the field of distributed systems, that of maintaining a global time and also of recognising data that is correct when multiple copies exists within the system.

#### **4.1.1.13 Connection bandwidth**

Connection bandwidth in a P2P system (at the logical network level) is dependent upon the available bandwidth at the physical network level, because although the logical network infrastructure can be seen as separate from the physical connections involved, it obviously still uses them. Although a (logically) direct connection could be made between two nodes on a P2P network, it may involve routing through many traditional routers and networks, so there is a bandwidth dependency upon the interconnected nodes/networks at the physical level as well. This is not entirely true however of P2P systems that exist upon the same

subnet, since it would then be the case of bandwidth limitations upon the utilised LAN and also that of the nodes on the P2P network.

Because direct communication is established between all nodes at the logical level, no routing is required and so nodes can only use the bandwidth that they themselves possess (discounting the physical network). Ultimately this makes this type of architecture as efficient (if not more so) as semi-centralised architectures. For example, two nodes with large bandwidth connections will always communicate with each other at the full potential. If communication had to be routed via a node with a low bandwidth connection then the efficiency of the connection between the nodes would drop.

There is of course much debate here concerning the impact of physical level networks influencing the bandwidth of logical level networks. The main point being stressed is that if more routing is to be performed at the logical level, it will very likely be mirrored at the physical level, and thus bandwidth will drop overall.

#### **4.1.1.14 Intermittent node connectivity**

Although all architectures will be affected to some degree by connectivity intermissions (at the logical network level), because nodes within direct communication architectures are connecting directly to one another this affect can be reduced. With indirect communication architectures, communication between nodes typically involves routing through other nodes and should a routing node become disconnected, this can have a significant effect on node communications. With direct communication architectures routing (at the logical level) is not required and so this situation will not arise.

#### **4.1.1.15 Peer Discovery**

Peer discovery normally hinges on the first few peers that are to be found. If a node cannot find any peers that are connected to the main congregation of peers, then that node will be out on a limb. Luckily this architecture can avoid the problems associated with peer discovery, because all nodes must be known for this architecture to work. This of course assumes that the network is not of a considerable size, which it is unlikely to be if using this type of architecture.

#### **4.1.1.16 Anonymity**

There are two dimensions to anonymity, that of the user and that of the node. User anonymity can be achieved at the application level so it is the location anonymity that will be looked at here.

Node anonymity typically involves the decoupling of node's IP address from that of the P2P application that it is running. This architecture as with all decentralised architectures must utilise network protocols such as IP addressing that already exist, this is because of the lack

of centralised points that can act as a bridge between the IP system and whatever addressing system the applications use to communicate. Anonymity would therefore be best implemented on a semi-centralised architecture that could facilitate such a bridging.

#### **4.1.1.17 Node addressing**

Node addressing within this architecture would most likely have to depend on IP addresses. As there is no control node that can oversee an alternative addressing mechanism, individual nodes would have to rely on maintaining an up-to-date list of node IP addresses themselves. The main disadvantage of relying on IP addresses is that it is practically impossible to tackle dynamic IP addresses and users who happen to frequently change between computers. Instead of an address being assigned to a particular user, an address would be assigned to each machine that could connect to the network.

This architecture is most likely to be chosen for a specialised purpose, and the network size would probably remain small scale using fixed node locations. The problems of node addressing that can appear in the other decentralised architectures may not have such an effect on this one.

If an alternative node addressing mechanism were required, then a semi-centralised architecture would be more suitable.

#### **4.1.1.18 Load balancing**

Whereas it can be difficult to achieve load balancing in indirect communication architectures due to nodes not necessarily being aware of all other nodes, it might be possible to achieve a degree of load balancing within direct communication architectures. Because each node is connected to every other node on the network, it would be possible for a node to be aware of what resources are available throughout the network. The problem would then be to attempt to orchestrate load balancing without the use of a central server. Such a method could involve the use of a logical token ring topology, this ring could be set up automatically by organising 'other node' lists on each node into some ordering (e.g. sorted by Network Interface Card (NIC) address).

#### **4.1.1.19 Manageability**

Managing a P2P network that uses this architecture may or may not be a problem, depending on how much automation is to be used in the management process and also the size of the network. As previously mentioned, this architecture cannot scale that efficiently and so it is likely to be used for P2P systems requiring small-scale specialised networks. Because of this the network may possibly be managed manually or with a mixture of manual and automatic management.

#### **4.1.1.20      Adaptability**

Due to the fact that each node within a direct communication architecture is in contact (at the logical network level) with every other node, the architecture should only need to adapt to node additions or removals from the network. Adaptations that are influenced by issues such as connection bandwidth or load balancing would not normally be an issue with this type of architecture.

However because the architecture is so rigid in nature (due to fact that each node has to know the address of every other node) adapting to changes might be difficult for this architecture. Should a node be added or removed, then every node on the network will need to be informed about it so they can update their node addressing cache.

#### **4.1.1.21      Repairability**

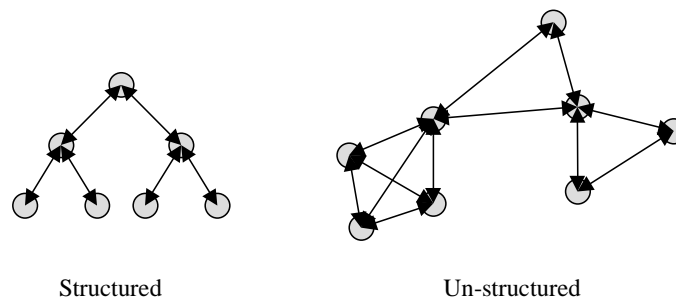
Providing repairability in a system based on this architecture can be problematic due to the lack of centralised control. This can make it difficult to detect defects within the system should they occur and to, consequently, make repairs. It is likely that a system using a direct communication architecture would have to rely on the owners of the nodes or, if they happened to be structured in some manner (for example, into clusters), some kind of local administrator identifying and rectifying the situation. As it cannot be presumed that any defects within the network will be detected and repaired, the situation could arise where a system is operating whilst still possessing defective nodes.

However, as previously pointed out, because this particular architecture has scalability constraints it may well be small enough to provide a degree of repairability without too many difficulties.

### 4.1.2 Indirect communication architectures (b + c)

This section discusses the implications the indirect communication architectures have on the properties that were identified in section 3. The two architectures have been grouped as in most cases they have similar effects on the properties.

An indirect communication architecture is one where all the nodes within the network can communicate with each other but often via another node. All the nodes are regarded as being equal and autonomous, and no single node exhibits control over the network. A structured indirect communication architecture is where some form of structure has been enforced onto the network (for example hierarchical). An unstructured indirect communication architecture is where no structure has been enforced. These architectures are illustrated in Figure 4.2.



**Figure 4.2** – Indirect communication architectures

#### 4.1.2.1 Reliability

An indirect communication architecture provides both advantages and disadvantages in terms of achieving a reliable system. The main disadvantage of such an architecture is that it can suffer from poor Quality of Service, in that, whilst in use the responsiveness of the system can vary considerably depending on the loads put on the system. Because there is no point of control within the network it means that the tasks of node discovery and communication is dependent on the network as a whole. Given the unpredictable and varied nature of a P2P network, the time it can take to achieve such tasks can vary from near instantaneous to not at all. This can have an apparent effect on the perceived reliability of the system.

However the perceived reliability of such an architecture can benefit from enhanced availability that can occur as a result of the redundancy inherent within such an approach. If an existing communication route between nodes fails then it is possible for a P2P system to adapt and to find an alternative. Similarly, various nodes can maintain copies of data used within the system and so act as an alternative sources should a node fail. Consequently, should failures occur within the system, it may still appear to be reliable to the user as alternative aspects of the architecture are being drawn upon.

#### **4.1.2.2 Scalability**

Both the indirect communication architectures are more flexible than their direct communication counterpart and this can make them much more scalable. The main disadvantage of these architectures is that some communication between nodes will have to be routed via other nodes of the network. In a very large scale P2P system this routing could result in a significant drop in quality of service (for example, if a message had to be relayed via 10 nodes before it reached its destination).

#### **4.1.2.3 Security**

There are advantages and disadvantages with respect to the provision of network security when adopting an indirect communication architecture. A main disadvantage is the difficulty in supporting the certification of nodes with the network. This can be even more problematic should the nodes require to remain anonymous to the rest of the network. In this respect decentralised architectures in general suffer when compared with semi-centralised alternatives that can offer a single certification server.

However, the lack of central foci also helps to make indirect communication P2P systems more resistible to network attacks. As no single node plays a key role within the system it becomes harder to bring the system down using methods such as denial of service.

#### **4.1.2.4 Survivability**

As with direct communication architectures, there are both advantages and disadvantages of achieving survivability within indirect communication architectures.

Survivability itself, draws upon other architectural properties such as security and fault tolerance. These all, to varying extents, can be best achieved when used within a semi-centralised architecture due to their general requirement for central points of control. As a consequence, generally for achieving survivability within a P2P architecture it would be most beneficial to use a semi-centralised architecture.

However, decentralised architectures do not rely on any single node (or nodes) within the network. This can make them less susceptible to some forms of attack, but also can make it harder to recover from an attack should one be successful. In addition, the inherent redundancy in P2P systems also means that should a node fail or be attacked, it can still be possible for the system to function normally.

Achieving survivability in indirect communication architectures is likely to be harder than with direct communication architectures as it is not the case that a single node will know every other node on the network. This makes it difficult for a node to be aware of what is happening within the network.

#### **4.1.2.5 Safety**

Indirect communication architectures are less suitable for systems where safety plays an important part due to the fact that there is no node that maintains any form of control over the system. For a safety system this is usually crucial, as the system needs to be constantly monitored for failures. Because of this, if a safety critical system were to be developed a semi-centralised architecture would be more suitable.

#### **4.1.2.6 Maintainability**

Indirect communication architectures can be difficult to maintain due to the lack of an overseer that can distribute and track updates to the system. Consequently maintaining the individual nodes is likely to be the responsibility of the node owner, or, if the nodes happened to be structured in some manner (for example, into clusters), a local administrator (person). As it cannot be presumed that these updates will be performed, and because it is impractical to enforce node updates, an indirect communication P2P system would need to be able to support legacy versions of the software.

#### **4.1.2.7 Responsiveness**

Indirect communication architectures typically rely on the discovery service to locate nodes on the network, and routing to support the communication between nodes. Both of these depend heavily on, what is typically, a very unpredictable and varied network. It cannot be guaranteed that a node will exist in a network at a given time, or that it will possess a high bandwidth connection. Consequently these factors can affect and reduce the responsiveness of an indirect communication P2P system.

#### **4.1.2.8 Responsibility, accountability and reputation**

Encouraging responsibility and accountability, and developing mechanisms to build and monitor a nodes reputation can be difficult with indirect communication architectures due to the fact that there is not a central node that oversees these tasks. In order to achieve these, an independent administrator would likely be required that could monitor and enforce rules within the peer network. A semi-centralised architecture would be the most suitable for achieving this.

#### **4.1.2.9 Availability**

Given the unpredictable and varied nature of P2P networks it is difficult to predict the availability of a P2P system at a given point in time. Furthermore, an indirect communication architecture has the disadvantage of not possessing the means to monitor the networks status, which is possible with a server node in a semi-centralised architecture.

The advantage of indirect communication architectures in terms of availability, however, is that if a failure should occur within the network, there is a good chance that the system will still be available (at least for unstructured indirect communication architectures). Communication can be re-routed via different nodes and there is a good chance that any information held on the failed node, would be duplicated elsewhere. A structured approach can still become disconnected and can result in a separation in the network; in this case may still be possible for the network to operate as two individual ones, until of course they can be married back together.

#### **4.1.2.10 Fault Tolerance**

Achieving fault tolerance within indirect communication architectures can be a difficult task due to the lack of central foci points that can monitor the status of the network. Should a fault occur in an indirect communication architecture, it is only going to become apparent when an attempt is made to either contact a node or use the network connection (depending on what has failed). Similarly, it is also difficult to assess the damage that has been caused by the fault.

Recovering from a fault within an indirect communication P2P system can also be a difficult task as it is hard to ensure that every node has reverted back to a suitable safe state. A possibility would be for a global alert to be sent out when a fault is discovered that can ensure that nodes revert to a safe state. However there is no guarantee that all nodes will receive this alert, and due to the lack of central control the rest of the network is not guaranteed to be informed should the fault be later fixed.

Repairing the fault is again difficult and would likely involve the person responsible for the node or network connection having to make the repair themselves. As this is not monitored, it cannot be assumed that this will be done and done properly.

Of the two indirect communication architectures, the unstructured approach is the more tolerant due to the fact that a node can be connected to the rest of the network via a variety of routes. In contrast a more structured architecture (for example a hierarchy) is likely to have limited number of connections (possibly even just one, to a node higher in a hierarchy) and so can be more prone to faults.

However, the main advantage for this architecture is that it does not have the points of failure that exist within semi-centralised architectures. For these architectures a fault occurring to a server node is likely to have a far greater impact on the system, than should a fault occur within a node in a decentralised architecture. From this respect, the decentralised architectures can be more fault tolerant.

Generally, for establishing fault tolerance within a P2P system, an indirect communication architecture is not ideal (apart from the lack of central points of failure) and a semi-centralised architecture would be the better option.

#### **4.1.2.11 Political and Legal Resistibility**

Compared to semi-centralised architectures, indirect communication P2P systems are likely to be much more difficult to shut down. It can be difficult to track down the nodes that exist within the network, and depending on the scale of the system, impractical to try and shut them down.

If ease of shutting down were a requirement for the system then utilising a semi-centralised architecture would be more suitable.

#### **4.1.2.12 Data integrity**

Data integrity within P2P systems is typically difficult to achieve and even more so in those that adopt indirect communication architectures. Data can be frequently passing between nodes and with no notion of a central repository, it can be difficult to achieve concurrency. With an indirect communication architecture it is highly likely that over time data that is held by a node, despite the node owner believing otherwise, will become invalid.

However, although an indirect communication architecture can suffer from concurrency issues, the fact that data redundancy is likely to occur within the system means that a form of inbuilt data backup can occur. Should the data become corrupt in one part of the system, then there is the possibility of retrieving it from another part.

#### **4.1.2.13 Connection bandwidth**

Connection bandwidth is a problem that will be encountered no matter what logical P2P architecture you use. With indirect communication architectures it can be particularly problematic in instances where the communications between nodes are routed through a low bandwidth (physical) connection. This could ultimately result in areas of the network playing a reduced role within the system.

However, indirect communication architectures (particularly those that are unstructured) have the advantage in that they are particularly adaptable, more so than their semi-centralised counterparts. Given that P2P networks are typically constantly changing, overtime the nodes and communication routes within the network can realign themselves to make full benefit of the high bandwidth connections and rely less on the low bandwidth connections. This has been seen to happen with existing P2P applications, such as Gnutella, in which a node can break a link with another node if it possesses a low bandwidth connection.

#### **4.1.2.14 Intermittent node connectivity**

Intermittent node connectivity is another property that applies to all P2P architectures. The main concern for indirect communication architectures are the effect they can have on

communication routing, the mechanism for discovering nodes and the awareness of which nodes are currently located within the network.

The disconnection of a node can invalidate an existing communication route between nodes and hinder the use of node discovery mechanisms. This means that alternative communication routes need to be discovered and this can have an effect on the overall performance of the system. The adaptable nature of indirect communication architectures, however, means that establishing a new route is typically not a significant problem.

Intermittent node connectivity can be a concern when trying to establish an awareness of which nodes are currently connected to the network. A disconnected node would typically leave the network without warning, and its absence would only be noted when another node tries to communicate with it. For indirect communication architectures to establish a relatively up to date picture of the network it would likely require the frequent launching of node discovery mechanisms. However such activities can be a drain on a nodes' resources.

#### **4.1.2.15 Peer Discovery**

Peer discovery in an indirect communication architecture would typically involve making use of a node discovery mechanism. This would operate by sending a communication from one node to another that would request details of all the nodes that it is aware of. The communication would in turn be then sent to these nodes. Details of known nodes would be relayed back to the originator.

A disadvantage of peer discovery in indirect communication architectures is that it is reliant on the nodes within the network being active, and also on the network connections between the nodes. Furthermore, should the network change there is no way of knowing this until another node discovery operation is launched. This can mean that what a node is aware of can become out of date and result in inaccurate information being fed back to the originator of the request.

Consequently, given these factors, it is difficult to predict the reliability of the node discovery service. If a fast and efficient peer discovery were required, a semi-centralised architecture would be the more appropriate choice.

#### **4.1.2.16 Anonymity**

Anonymity within indirect communication architectures is prone to the same difficulties as with direct communication architectures.

Node anonymity typically involves the decoupling of a node's IP address from that of the P2P application that it is running. These architectures as with all decentralised architectures must utilise network protocols (such as IP addressing) that already exist. This is because of the lack of centralised points that can act as a bridge between the IP system and whatever addressing system the applications use to communicate. Anonymity would therefore be best implemented on a semi-centralised architecture that could facilitate such a bridging.

#### **4.1.2.17 Node addressing**

Node addressing within indirect communication architectures would most likely have to depend on IP addresses. As there is no control node that can oversee an alternative addressing mechanism, individual nodes would have to rely on maintaining an up-to-date list of node IP addresses themselves. The main disadvantage of relying on IP addresses is that it can be difficult to tackle dynamic IP addresses and users who happen to frequently change between computers. Instead of an address being assigned to a particular user, an address would be assigned to each machine that could connect to the network.

Of the two indirect communication architectures, the structured approach is likely to be better equipped for supporting node addressing. By structuring the nodes it is possible to group nodes together and assign them similar addresses. This is in a similar vein in which IP addresses within DNS work.

If an alternative node addressing mechanism were required, then a semi-centralised architecture would be a more suitable option.

#### **4.1.2.18 Load balancing**

Achieving load balancing within an indirect communication architecture can be difficult as there is no central node that is aware of what resources are available on the network. Consequently you can have communication routes established that are built over low bandwidth connections, or draw heavily on a node that has little processing power.

However, despite the lack of logical network awareness, indirect communication P2P systems have the advantage of being particularly flexible and can adapt to suit the available network resources. The nodes with the most networking capabilities will typically move to the centre of the network, and those with the least will move out to the edges (all within the scope of the logical network). This has been shown to happen with the Gnutella P2P system (chapter 8 of [6]).

#### **4.1.2.19 Manageability**

Systems that are based on indirect communication architectures can be particularly difficult to manage due to the lack of one or more control nodes. As each node within the network essentially acts independently it is very hard to track what is happening within the network.

If it were desired to have a P2P system that is manageable then a semi-centralised architecture would likely be a more suitable choice.

#### **4.1.2.20      Adaptability**

Indirect communication architectures, and in particular those that are unstructured, have the potential to be the most adaptive of all the P2P architectures discussed in this document. Because nodes do not have to be focused around central nodes nor have to be aware of every other node on the network, these types of architectures can easily adapt to changes within the network topology without requiring any form of management. This type of network can adapt to issues such as making more efficient use of the available bandwidth or to improving load balancing. Gnutella is an example system where this has been known to occur.

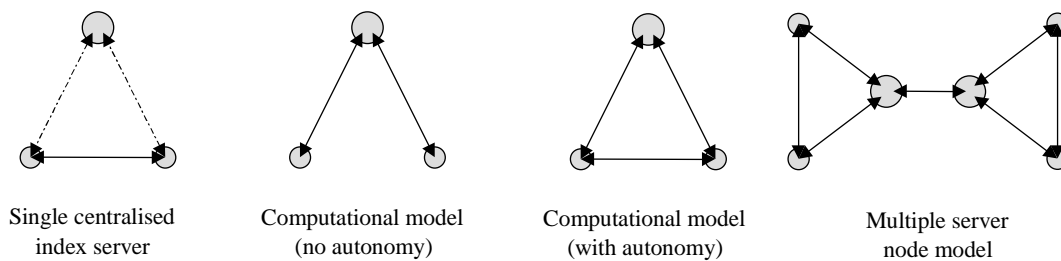
#### **4.1.2.21      Repairability**

As with direct communication architectures, achieving repairability in indirect communication architectures can be difficult due to the lack of central points of control. This can make it difficult to detect defects within the system should they occur and to, consequently, make repairs. It is likely that systems using an indirect communication architecture would have to rely on the owners of the nodes or, if they happened to be structured in some manner (for example, into clusters), some kind of local administrator identifying and rectifying the situation. As it cannot be presumed that any defects within the network will be detected and repaired, the situation could arise where a system is operating whilst still possessing defective nodes.

## 4.2. Semi-centralised Architectures

This section focuses on the semi-centralised architectures. Unlike decentralised architectures, the different semi-centralised architectures make no significant difference to the individual properties. Consequently this section discusses the impact of semi-centralised architectures on the properties as a whole.

Semi-centralised architectures are architectures that possess one or more central nodes. These nodes act as a focal point for the network and in some cases can maintain control over the network as a whole (computational models). Where there is communication between nodes, this is normally established directly with the central nodes acting as a facilitator (providing the destination nodes address). Figure 4.3 illustrates the semi-centralised architectures.



**Figure 4.3** – Semi-centralised architectures

### 4.2.1 Reliability

Semi-centralised architectures provide both advantages and disadvantages in terms of achieving a reliable system. In contrast to decentralised systems, semi-centralised architectures do not have to rely on a node discovery service as a server node can instead maintain a look up table. Additionally, once a node has been located a direct communication link can normally be set up, removing the need to have messages routed through the logical network. Both these factors can improve the quality of service of the system.

Reliability can also benefit from the inherent redundancy of P2P systems. With semi-centralised architectures this can involve the fact that data can be replicated among the client nodes. Should one node become disconnected from the network, the data can be obtained from alternative locations. Napster is an example system in which this occurs. Furthermore, there is no reason why this replication cannot be extended to the server nodes. This can strengthen the reliability of the system by providing backups to the server nodes and thus removing the single point of failure.

However, a drawback of semi-centralised architectures is their reliance on the server nodes, which essentially can act as single points of failure within the system. Should the server be disconnected from the network then the performance of the system as a whole can be seriously affected. This is particularly the case for the two computational architectures where client nodes depend heavily on the server nodes.

#### **4.2.2 Scalability**

Scalability is less of a problem for semi-centralised architectures than for some decentralised approaches. Nodes typically only have to know the server node in order to make use of the system. The danger, however, is that as the network increases with size the server nodes can become overloaded. In order to create large scale semi-centralised systems, it is likely that mechanisms will need to be considered to ensure the server nodes are not overloaded.

#### **4.2.3 Security**

There are advantages and disadvantages with respect to the provision of security when adopting a semi-centralised architecture and these are typically the inverse of those exhibited by decentralised architectures. The use of server nodes means that it is relatively simple to establish certification within a system, whilst at the same time being able to support the anonymity of users.

A disadvantage of semi-centralised architectures is that the server nodes become obvious points of attack (denial of service attacks being an example). Consequently, server nodes will need to be made more resistible to attack.

#### **4.2.4 Survivability**

Semi-centralised architectures can provide certain advantages for when trying to maximise a system's survivability. As well as the benefits of redundancy that is inherent within in all P2P architectures, semi-centralised architectures also possess central nodes that can monitor the network. These central nodes can assist in achieving greater security, fault tolerance and hence survivability within a system.

The main disadvantage is that the central nodes themselves become the obvious targets for attack. Should an attack succeed, or a fault occurs, a semi-centralised based system has the potential to become severely crippled. Consequently, it is a priority to ensure a high degree of survivability for the central nodes within the network. Of course survivability can also be improved by increased the number of centralised nodes that are used.

#### **4.2.5 Safety**

Semi-centralised architectures are more suitable for safety critical systems than the decentralised alternatives. Because one or more nodes within the architecture can maintain a degree of control, it is easier for the system to be monitored for any faults that may occur. This is much more difficult to achieve with decentralised architectures.

#### **4.2.6 Maintainability**

Semi-centralised architectures are likely to be easier to maintain due to the use of central nodes. These nodes can distribute and monitor any updates that are required for the system. However, if updates are to be applied manually (i.e. by the owner of each node), it still cannot be presumed that these updates will be performed. Semi-centralised architectures do allow for the possibility of automatic upgrades to the system however.

#### **4.2.7 Responsiveness**

Semi-centralised architectures do not have to rely on a discovery service to locate nodes or on routing to support the communications between nodes, both which are prominent in most decentralised architectures (with the exception of those that use a direct communication architecture). The locating of nodes can be done via a central index node, and direct communication between the nodes can then be set up. By reducing the dependence on what is typically, a very unpredictable and varied network, a semi-centralised systems responsiveness should generally be higher than an equivalent decentralised system (except for systems based on direct communication architectures). However, should the central nodes fail this can have a severe effect on the responsiveness of the system.

#### **4.2.8 Responsibility, accountability and reputation**

It is likely to be easier to encourage responsibility and accountability, and to develop mechanisms to build and monitor a node's reputation with semi-centralised architectures than with decentralised architectures. With semi-centralised architectures, the central nodes within the network can be used to monitor and enforce rules, and also to support the use of reputation tracking and micro payment techniques. This is can be more difficult to achieve with decentralised architectures.

#### **4.2.9 Availability**

As with all P2P networks it can be difficult to predict the availability of a P2P application at a given point in time due to their unpredictable nature. However, semi-centralised architectures possess both advantages and disadvantages with respect to system availability when compared with decentralised alternatives.

In a semi-centralised architecture it can be much easier to monitor the status of the network. Should a node fail then the central node can relay this information to all other nodes on the network. However, semi-centralised architectures typically rely significantly on these central nodes and should they themselves fail then in all likelihood a significant part of the systems functionality will be lost and hence reducing availability.

#### **4.2.10 Fault Tolerance**

With semi-centralised architectures it is possible for the central nodes to monitor the peer network for faults. Should a fault occur it can be identified relatively quickly, the damage assessed and the rest of the network notified.

Due to the distributed nature of P2P systems repairing a fault can still be difficult, however with semi-centralised architectures the process can be monitored or even carried out by a central node. For more freely deployed systems fixing the fault would likely involve the person responsible for the node or network connection having to make the repair themselves. A central node, however, could still monitor this.

A major disadvantage for fault tolerance in a semi-centralised architecture is the reliance on the central nodes. Should a fault occur in one of these, then the effects on the system are likely to be significantly more serious than should one of the edge nodes fail. Consequently, it becomes particularly important to make the central nodes fault tolerant.

For establishing fault tolerance within a P2P system, semi-centralised architectures provide advantages and disadvantages, however they are possibly a more viable option than the decentralised alternatives.

#### **4.2.11 Political and Legal Resistibility**

Compared to decentralised architectures, semi-centralised P2P systems can be relatively easy to shut down due to their reliance on the central nodes. Should these be removed from the network, the functionality of the system would likely be partially or even totally impaired.

If ease of shutting down were a requirement for the system then a semi-centralised architecture would be the more suitable option.

#### **4.2.12 Data integrity**

Data integrity within P2P systems can be difficult to achieve, however semi-centralised architectures arguably provide the best mechanisms for accomplishing this. Unlike decentralised architectures, the central nodes within the network can monitor where data exists and ensure a degree of concurrency. Should data become invalid on one node then a central node can bring this to the node users attention.

Semi-centralised architectures also possess the advantage that exists with decentralised architectures, that of data redundancy. Should the data become corrupt in one part of the system, then there is the possibility of retrieving it from another part. If the data was stored centrally, on the central nodes, then this would be more of a problem.

#### **4.2.13 Connection bandwidth**

Connection bandwidth is a problem that will be encountered no matter what P2P architecture you use. Semi-centralised architectures have the advantage over decentralised alternatives, in that direct communication between nodes is typically used once the address of the destination node has been identified. This removes the need for routing and also the possibility of this routing being performed over low bandwidth connections.

However, generally semi-centralised architectures are not as adaptable as some decentralised architectures. Although the addition of new nodes to the network would not be a problem, a node that is a long distance away from a central node and can only make use of a low bandwidth connection, would not benefit from decentralised routing that might make use of a higher bandwidth connection to reach the destination.

#### **4.2.14 Intermittent node connectivity**

Intermittent node connectivity is another property that applies to all P2P architectures. Although it has less effect on semi-centralised architectures than on decentralised ones (where it effects routing, node discovery and awareness), it can still cause problems.

An advantage of a semi-centralised architecture is that the central nodes can monitor the network and establish an awareness of which nodes are currently connected (ICQ is an example of this). Should a node be disconnected, then the rest of the network can be notified. However, node disconnection can still cause problems especially if another node was making use of its resources at the time of disconnection. In these circumstances, the inherent redundancy of P2P systems in terms of multiple copies of data and resources can prove to be beneficial.

The main concern for semi-centralised architectures is if the central nodes themselves are disconnected. In such circumstances the operation of the system can be severely impaired.

#### **4.2.15 Peer Discovery**

Peer discovery in semi-centralised architectures typically involves making use of the central nodes as an indexing mechanism. When a node connects to the network it would register with the central node. When a node wishes to communicate with another node in the network, it query's the central node for the destination nodes address. Once this is obtained, direct communication between the two nodes can be established.

The main concern with peer discovery in semi-centralised architectures is that it is reliant on the central nodes. Should the central nodes fail, then the peer discovery mechanism will also fail. Consequently, in semi-centralised architectures it can be beneficial to have a decentralised peer discovery mechanism as a backup should such a failure occur.

Semi-centralised peer discovery has advantages over decentralised mechanisms in that it is more efficient and more up-to-date (the central node is more aware of the current state of the

network). Consequently, if a fast and efficient peer discovery were required, a semi-centralised architecture would be a more appropriate solution.

#### **4.2.16 Anonymity**

Achieving anonymity within a P2P system is likely to be easiest to achieve when a semi-centralised architecture is used. Whereas with decentralised architectures it is difficult (if not impossible) to hide a nodes address from the rest of the network, with semi-centralised architectures it is possible that only the central nodes need to be informed. Furthermore, if an alternative addressing mechanism is used (for example user centric) then additional layers can exist between the user and their IP address.

Semi-centralised approaches to anonymity do however rely on the nodes registering with the central nodes. Although the amount of information provided can be up to the user, the fact that the central nodes store the information of all nodes on the network can pose a security risk. However semi-centralised approaches do make it easier to support some security issues, such as certification.

#### **4.2.17 Node addressing**

Node addressing within semi-centralised architectures is not as restricted as in decentralised architectures. Although an addressing system built around IP addresses can be used, the central nodes allow for more sophisticated addressing mechanisms. Such mechanisms could include user centric addressing (discussed in section 3.2.1) which is used in existing P2P systems such as ICQ. Node addressing that is orchestrated by a central node has advantages in that it can support anonymity and also provide better support for security aspects such as certification.

The disadvantage of this form of node addressing is that it places significant reliance on the central nodes. Should these fail then the addressing mechanism will also fail.

#### **4.2.18 Load balancing**

Achieving load balancing is likely to be easier within a semi-centralised architecture than with a decentralised architecture, due to the fact that the central nodes can monitor what resources are available on the network and designate load to them accordingly. This can reduce the chance of low bandwidth communication channels being heavily used, or nodes with low processing power being overworked.

The disadvantage of semi-centralised architectures is that they are fairly rigid, being centred around the central nodes, and do not have the ability to evolve like the decentralised alternatives. However, if a P2P network is well managed by the central nodes then this should not pose a significant problem.

#### **4.2.19 Manageability**

Manageability is likely to be easier to achieve with semi-centralised architectures than with decentralised architectures due to the fact that the central nodes can oversee the network.

Depending on the type of semi-centralised architecture used, the central nodes can have a varying influence on the system. Central nodes within the two computational architectures possess almost full control over the system, whereas index node architectures will typically only act as central points of focus for the network and take care of, for example, any discovery, reputation tracking and security issues.

Due to the difficulty in managing decentralised architectures, if being able to manage a P2P system were desirable then a semi-centralised architecture would likely be the most suitable.

#### **4.2.20 Adaptability**

Semi-centralised architectures may not appear to be as adaptable to change as their decentralised counterparts because to a certain extent the nodes on the network have to focus around the central nodes.

However, because communication between nodes is direct, the architecture would not necessarily have to adapt to tackle bandwidth issues. Furthermore, as only the central nodes need to be aware of which nodes exist in the network, adapting to the addition or removal of nodes can actually be simpler than with decentralised architectures. In the same way, central nodes can also be aware of what resources are available and balance load accordingly. In contrast to the more freeform nature of adaptations in indirect communication architectures, those occurring within semi-centralised architectures are likely to be more managed.

The main disadvantage of semi-centralised architectures is the fact that all nodes have to connect to one or more of the central nodes. If these nodes happen to be a large physical distance apart, then responsiveness/bandwidth issues could arise. Whereas with some decentralised architectures you could envision the creation (via adapting) of localised clusters of nodes, with semi-centralised architectures this is unlikely to happen unless many more central nodes are used.

#### **4.2.21 Repairability**

Repairability is likely to be easiest to achieve with semi-centralised architectures due to the fact that they possess central nodes. These nodes could monitor the peer network for defects and then either attempt to repair the fault automatically, or inform the owner of the peer so that it can be done manually. However, if repairs are to be done manually, it still cannot be presumed that these updates will be performed although it is easier to monitor.

### **4.3. Summary of analysis**

This section has analysed the effect the different network architectures that were identified in section 2, can have on the architectural properties that were identified in section 3.1. Table 4.1 provides a summary of this analysis.

	<b>Decentralised – Direct Communication</b>	<b>Decentralised – Indirect Communication</b>	<b>Semi-Centralised</b>
<b>Reliability</b>	Weak points in network architecture can be more easily identified. QoS can be more easily adhered to.	Can suffer from poor QoS. Could benefit from redundancy.	Reliance on server nodes is a drawback. Can benefit from redundancy.
<b>Scalability</b>	Serious problems with scalability.	More scalable than direct communication architecture. Communication routes being reliant on other nodes could be a problem. QoS can suffer from large-scale implementations.	Generally less problematic than decentralised architectures. Network size increases can put a strain on server nodes.
<b>Security</b>	Lack of centralised control seriously limits current security implementations.	Lack of central control limits security implementations. Could be more resistible to attacks.	Use of server nodes aids security implementation but become obvious points of attack/failure.
<b>Survivability</b>	Implementation dependent – depends on how it is engineered.	Can be hard to launch a successful attack against but can also be hard to recover from one.	Easier to attack due to existence of server nodes, but can be easier to discover an attack attempt and also recover from a successful attack.
<b>Safety</b>	Lack of centralised points of control makes it extremely hard for carrying out system safety checks. Not suitable for safety critical systems.	Same as for Decentralised – Direct Communication.	Better degree of safety control can be achieved than with decentralised architectures.
<b>Maintainability</b>	Because of scalability constraints, maintainability could be achieved by others means apart from automatically (manually or semi-manual)	Difficult due to lack of centralisation, maintenance is likely to be handled by node owners. Cannot guarantee that updates will be carried out	Centralisation means easier maintenance, possibly allowing automatic updates.

<b>Responsiveness</b>	Because it is a direct communication architecture responsiveness could be optimal.	Sub-optimal due to discovery and routing through third party nodes.	Not quite as good as in direct communication architectures as it has a discovery lookup overhead.
<b>Responsibility, accountability and reputation</b>	Lack of central points of control makes it hard to incorporate reputation tracking.	Difficult to achieve due to lack of central control.	Easier to achieve accountability and reputation tracking.
<b>Availability</b>	It is possible on small-scale networks but will likely incur a networking overhead.	No direct way of monitoring the network status. Large-scale networks have the advantage of being highly available due to resilience from attacks and replication.	Easier to monitor network status by using monitor nodes. These monitor nodes provide a weak link in the system though.
<b>Fault tolerance</b>	Can support global alert to let others know of faults, unless the fault lies in the communication infrastructure.	Not plausible to use global alerts for large-scale systems.	Can handle network monitoring through server nodes. Again these nodes are the weak point. On the whole this is a more viable option.
<b>Political and legal resistibility</b>	Not really a problem as the architecture is likely to be use for internal business systems.	Difficult to shut down and possibly difficult to track down nodes on large-scale systems.	Much easier to shut down due to the inclusion of server nodes.
<b>Data integrity</b>	Can suffer from problems of maintaining global time and identifying incorrect data when multiple copies exist.	Same as for Decentralised – Direct Communication.	Centralisation can provide the best mechanisms for achieving data integrity, as it is better equipped for handling the problems faced by decentralisation.

<b>Connection bandwidth</b>	Because direct communication is established between all nodes it is possible to make the most effective use of bandwidth	Problematic due to the fact that communications may well be routed through third party nodes with a low bandwidth. To some degree the network is able to adapt itself to compensate for this.	Direct communication is achieved once node discoveries are performed through the use of server nodes. Not as adaptable as decentralised indirect architectures.
<b>Intermittent node connectivity</b>	Because routing in this architecture can be minimised intermittent node connectivity may have little effect, except when trying to contact a node that has gone offline.	Due to the reliance of third party nodes for routing this architecture is most likely to be affected by intermittent node connectivity.	Can suffer from server nodes being disconnected.
<b>Peer discovery</b>	Not really affected, as any node should already know all other nodes.	Problematic due to the fact that very little is known about the network when a node connects to it. It must discover neighbours for itself and this discovery is liable to be routed through nodes that may go offline.	Central server nodes index the locations of all connected nodes. These central server nodes are the weak link though.
<b>Anonymity</b>	Not the best way to decouple IP addresses from that of a node. Traditional network protocols must be used to communicate with others.	Same as for Decentralised – Direct Communication.	Best for providing anonymity as the server nodes can act as a bridge to decouple the IP address from users/nodes.
<b>Node addressing</b>	Likely to use fixed IP addresses so this may not be a problem.	Difficult due to dynamic IP addresses being used, hard to keep nodes up to date with changes.	Indexes on server nodes are the weak points.
<b>Load balancing</b>	May be possible due to the fact that a logical network (logical token ring) could be produced.	No real way of effectively monitoring load on the network and locating bottlenecks.	Capable of network monitoring but the network itself is fairly rigid.

<b>Manageability</b>	Possible to be managed automatically or manually, but unlikely that it can be managed automatically.	Hard to manage and track what is happening in the network.	Better suited to managing itself.
<b>Adaptability</b>	Very rigid and not well suited to adaptations.	Extremely fluid environment where adaptations are fairly hard to stop.	Easy to adapt, but in a more managed manner
<b>Repairability</b>	Difficult to monitor network for faults and consequently make repairs due to the lack of coordinator nodes.	Even harder to monitor network for faults and consequently make repairs due to the lack of coordinator nodes and the fact that each node does not know every other node within the network.	Easier to achieve due to the fact they can possess coordinator nodes.

**Table 4.1** – Summary of the architectural property analysis

## 5. Discussion

### 5.1. Transactions, concurrency, and recovery

An important aspect of any multi-user system is its ability to cope with concurrent transactions and to ensure that accesses made by one user does not interfere with another. Because of the importance of ensuring correctness during transactions a transaction should have certain properties; -

1. Atomicity – A transaction must be completely fulfilled or not at all.
2. Consistency – A transaction must take the system from one consistent state to another.
3. Isolation – Each transaction must be performed without any interference from other transactions.
4. Durability – Upon completion of a transaction, all effects of the transaction must be preserved in permanent storage.

If a transaction fails at any point the system should be able to restore any state that was in effect before the transaction was called, that is it should recover from failures, even when the failure is due to a total system crash.

With P2P applications, the ways in which transactions and recovery are carried out will typically depend upon the form of system that is in use. The ‘distributed systems’ and ‘database’ communities have ample knowledge of transaction-based systems and should be used as a reference point for developers. [11]

Semi-centralised architectures are more suited to major transaction-based systems; this is due to the fact that the servers within such a system can help co-ordinate efforts between peers. Peers would generally be communicating without going through any server (apart from discovery), yet the server could still prove useful for tasks like keeping a temporary backup of transaction orderings and could even negotiate critical transactions between different peers once the ordering and any other issues made by the peers have been resolved. This does not mean that the P2P system is turning into a client\server model, it simply means that it is a server enhanced P2P system as the main transactions are still between peers.

Fully decentralised systems may well require a re-think on how to implement transactional based systems because of the wide and varied distribution of resources and the lack of any central points of control. Messages in a decentralised system are not guaranteed to arrive; this would have a serious impact on the completion of transactions. Another concern with decentralised systems is the fact that broadcast/multicast messages are quite often used. If these happen to be transaction based then there may well be problems with multiple copies of messages propagating around the network.

Things get more complicated when combining architectures into a more complex system. Problems that normally do not arise, or can easily be handled, within a particular architecture

may prove troublesome once architectures are merged together and it also becomes harder to envision what could go wrong.

Recovery from transaction failures is likely to be easier to accomplish with semi-centralised architectures, due to the fact that servers on the network can maintain a level of awareness. Decentralised systems on the other hand would require careful consideration as to when and how to implement a recovery procedure, it may well be possible to negotiate that one of the participating peers be used to log transactions and keep a watchful eye on things.

Atomic commit protocols can allow peers on a network to orchestrate transactions and also recover from aborted transactions. Typically a peer would be responsible for the serialised access to any of its objects but there could also be a requirement for ensuring that transactions are serialised globally within the network. Careful use must be made of locking schemes here since there would typically be a high risk of distributed deadlocks.

Transaction based systems typically require a tightly coupled message passing system to support the serialisation of transactions, ensure the correctness of operations and the final results of the operations. P2P systems however, are generally loosely coupled, so for them to support serialised transactions it may require developers to alter the way they think about current transaction based systems and modify current beliefs about transactions to fit the P2P paradigm.

Although this needs to be more properly researched and analysed, there are several obvious areas of concern if transactions are to be supported by a P2P system; -

1. Does the system allow a global awareness for when transactions are to be finally committed?
2. Can the system ensure that all transactions have been fully committed correctly by all nodes concerned?
3. Can the system support a global rollback and ensure that all nodes involved in the transaction perform their respective rollback actions correctly?
4. How do multiple concurrent transactions occurring between nodes in the network affect each other?
5. If it is possible for errors to occur how can it be ensured that they don't affect the validity of information within the system?

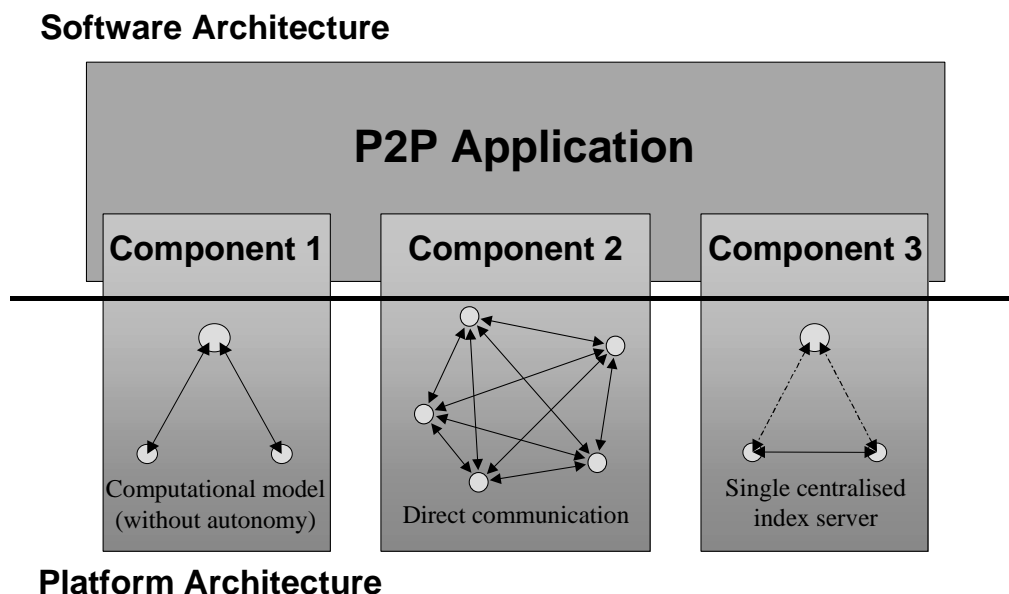
Although a lot of work have been covered in these areas in distributed system environments, it may not suffice for ad-hoc systems such as P2P, where nothing can be absolutely guaranteed as certain. And creating a tightly coupled P2P system to accommodate transactions undermines the whole concept of P2P; after all it is the ad-hoc nature of P2P that makes it what it is.

## 5.2. Software Architectures

Up to now this document has examined the different logical network architectures that can be used within P2P systems. This section moves on to examine the software architectures that could be used. Software architectures are now an established means for describing the overall structure of a system and a framework for sub-system control and coordination. Within a P2P system the software architecture would be built on top of one or more logical architectures.

Existing P2P application developments have tended to adopt software architectures that on the whole map directly onto the logical architecture. For example, ICQ uses a software architecture that essentially maps directly onto a single centralised index server logical architecture. SETI@home uses a software architecture that maps onto a computational model logical architecture.

Large and complex software architectures are often broken down into components during design and development. For example, Enterprise Java Beans [13] are components that can be used within application development. If such an application was to be developed as a P2P application then the situation could arise where the different components benefit from using multiple different underlying logical architectures. For example, a component that requires lots of computational power could benefit from using a computational logical architecture, or a component that involves collaborative user activities may benefit from a index server logical architecture. Figure 5.1 illustrates this.



**Figure 5.1** - How software architectures could use multiple logical architectures

The possibility of applications being built on a hybrid of logical architectures can raise dependability issues that would not normally effect those that only build on top of a single architecture. Not only could the components within the application have an influence on each other, it could also be the case that a peer is used within multiple components and thus its operation could have less obvious influences. Possible issues include:

**Reliability issues** – two aspects 1) the failure of a component having an effect on another component, and 2) the failure of peer having an effect on more than one component.

For the former, the failure of a component to operate as expected could have a direct or indirect impact on other components within the application. Such a failure could be caused, for example, by a breakdown in the logical architecture it utilises.

With the latter, it could be the case that two or more components within an application *share* nodes in order to achieve their goals. For example, in figure 1 the same peer might take active roles in both component 1's computational architecture, and component 2's direct communication architecture. The sharing of nodes between components means that the failure of these nodes could have a far wider effect on the overall application.

**Concurrency issues** – if components are going to share nodes then there is also the danger of data concurrency if both components happen to manipulate the same data at the same time. Consequently it might be necessary to ensure that only one component can have control of the node at a given point in time, or at the very least, access to the data that is on the node.

**Loadbalancing issues** – similar to concurrency but ensuring that shared nodes are not overworked. If a node is shared between components then to avoid overloading, its role within the respective components may have to be reduced to compensate. Again it might be necessary to ensure that only one component can have control of the node at one time (something akin to process scheduling strategies used within multi-threaded operating systems).

**Security issues** – shared nodes could act as 'back doors' into another component. Should the logical architecture for one component be compromised, then a node from this component that is shared with another component may become a launching point for further attacks. To avoid this it might be necessary to use different security mechanisms for each component.

**Manageability issues** – if an application makes use of multiple logical architectures then it is likely to be harder to manage. It is likely that each logical architecture will have to be managed by the respective component, rather than globally.

**Node addressing issues** – if nodes are shared between components then it could be difficult to use a global addressing scheme especially if the logical architectures used are significantly different. In such circumstances it might be necessary to have a different node addressing mechanism for each component.

It cannot be assumed that large and complex P2P applications will operate best from utilising only one type of logical architecture. For some applications it might be more beneficial for that components that comprise it to adopt different logical architectures. By using a hybrid of architectures, dependability issues can become apparent that would not normally appear with a single logical architecture based application. It is likely that with such a hybrid application, the most important requirement would be to keep the interference between components down to a minimal. This would especially be the case if a peer nodes are shared between components.

## 6. Conclusions

This document has attempted to identify the network architectures, and their corresponding properties, that can be used as a basis for the development of P2P systems.

The document commenced by identifying the key network architectures and split these into decentralised and semi-centralised categories. Decentralised architectures represent those in which all nodes within the network are regarded to have equal status, semi-centralised architectures represent those in which there exists one or more nodes that perform authoritative roles within the network.

The document then proceeded to identify the properties that P2P architectures could possess, before finally analysing how the different architectures could have an effect on the architectural properties (Table 4.1).

Analysis not only reaffirmed the expected differences between decentralised and semi-centralised architectures, but also highlighted significant differences between the decentralised direct and indirect communication architectures. Although direct communication architectures are not feasible in large-scale environments, when used in smaller settings they can exhibit particularly advantageous characteristics such as high responsiveness and reliability.

The dependability of a P2P system relies to some extent on security, as it does in most systems. Of the two mainstream architectures that have been identified (decentralised and semi-centralised) it is apparent that the semi-centralised architecture can accommodate a higher level of security than that of the decentralised architectures. This is due to the fact that security is usually implemented with what is available and affordable at the time it is required. Current security tools and techniques have evolved from the security needs of traditional client-server architectures, which have closer ties to semi-centralised architectures than decentralised ones. Considering this and the fact that the re-emergence of peer-to-peer computing is so young, it will likely be some time for security techniques to emerge to aid implementations in this new field.

Many dependability requirements affect other requirements, sometimes adversely. This can happen when attempting to architect most software-based systems and is something that must be resolved at the time, usually at the cost of some requirements. This document has attempted to point out some of these contradictions so as to make the reader aware of their existence.

In general it was observed that if a significant amount of monitoring was required within a P2P system (for example, to achieve security, fault tolerance, maintainability, etc) then a semi-centralised architecture would be the more suitable.

It should be noted that only an initial analysis has been presented in this document. A great deal more experimentation is required in order to fully understand the affects the architectures can have on the dependability of a P2P system. Such experimentation would likely need to involve the testing of systems on a large scale.

Finally, this document has also not considered the possible dependability issues that could arise as a result of interoperability between different P2P protocols and development API's (i.e. JXTA, Gnutella, etc). The technology (P2P) is still too young to standardise to any particular protocol, and to do so would commit the P2P industry to a definition that may not suit forthcoming P2P applications. The dependability issues relating to interoperability can be explored at a later date when a clearer picture with regards to P2P standardisation has been developed.

## References

- [1] Napster. More information can be found at the URL <http://www.napster.com>
- [2] FreeNet. More information can be found at the URL <http://freenet.sourceforge.net>
- [3] Domain Name Service (DNS). More information can be found at the URL <http://www.rad.com/networks/1995/dns/dns.htm#doc1>
- [4] SETI@home. More information can be found at the URL <http://setiathome.ssl.berkeley.edu>
- [5] Ian Sommerville, *Software Engineering*, Addison Wesley, 2001
- [6] Andy Oram (editor), *Peer-to-Peer: Harnessing the power of Disruptive Technologies*, O'Reilly publishing, 2001
- [7] Ian Sommerville, *The DISCOS Method presentation*, Lancaster University
- [8] Ellison, R. J., Fisher, D. A., Linger, R. C., Lipson, H. F., Longtaff, T. A., Mead, N. R., SURVIVABILITY: Protecting Your Critical Systems, In *IEEE Internet Computing*, 3 (6), 55-63, Nov/Dec, 1999.
- [9] Gnutella. More information can be found at the URL <http://www.gnutella.com>
- [10] ICQ. More information can be found at the URL <http://www.icq.com>
- [11] Distributed Systems - Concepts and design, Coulouris, Dollimore and Kindberg, 3rd Edition, Addison Wesley, 2001
- [12] More information can be found at the URL <http://www.kb.cert.org/vuls/id/570167>
- [13] More information can be found at the URL <http://java.sun.com/products/ejb/index.html>

## **7. Appendix A – Common security risks**

This section looks at common security risks and gives a description of each, it also classifies each into where each one into a cause, hazard or accident. Causes are the events or happenings that can lead to a hazard or vulnerability, hazards and vulnerabilities then have the potential to lead to an accident or attack.

### **7.1. Causes**

These are common ways that can lead to a hazard or vulnerability occurring.

#### **7.1.1 IP Spoofing**

IP spoofing is typically part of an attack plan, and it has relations to trust. Once a client has connected to a server, or another client as in P2P, and has gone through some authentication challenge or trust check a third client (the perpetrator) could take over its identity, especially if identities are based around IP addresses. One scenario is a client that connects to a server and authenticates itself; the perpetrator then steps in and launches a DoS attack on the client, thus disabling it. Using IP spoofing the perpetrator then carries on the communication with the unsuspecting server. This is obviously an over simplification, as problems are encountered by the perpetrator (i.e. getting the packets sequence numbers correct so the server doesn't know anything's amiss), but going into that amount of detail is not required here.

In a P2P system each peer node typically has some form of unique identifier, which could act as a form of IP address. With this in mind it is not hard to imagine spoofing this identifier, as a result P2P systems may not be immune to spoofing.

#### **7.1.2 Packet sniffing**

Packet sniffing is essentially a form of wire tapping for networked computers, and is a particular problem with Ethernet based networks and cable networks. Network cards have a type of filter built in that stops computers from seeing traffic intended for others. It is possible though for this filter to be disabled, and this is one of the jobs that a packet sniffing application can do.

Now because a P2P system can be built using a logical network, peers connect to each other and transport data intended for other peers. Packet sniffing could be a problem on P2P networks.

### **7.1.3 Email spoofing**

When an email appears to have been sent by one source, when in fact it originated from another, is called email spoofing. It is used as a joke, spreading email viruses and even for corrupt social engineering.

## **7.2. Hazards and Vulnerabilities**

These are states that a system is in when a mishap has the potential to happen.

### **7.2.1 Trojan horse programs**

Unsuspecting users usually install Trojan horse programs unknowingly, typically through unsigned and un-trusted application code. They are a common way of gaining access to a user's computer without the users knowledge or consent. Typical detection and removal are through the use of virus scanners.

### **7.2.2 Remote administration and back door access**

These are more commonly found on windows based operating systems, 'Netbus', 'BackOrifice' and 'SubSeven' programs are highly popular, this form of malware runs as a service on a users computer and allows access to the computer through the use of a client application from a remote location. Typical detection and removal are through the use of virus scanners.

### **7.2.3 Email virus spreading**

Most media reports of mass viral infections, such as the 'Melissa virus', are email borne. Ending about six to eight years ago the most frequent forms of computer viruses infected the boot-sectors of floppy disks, this was a fairly reliable way of spreading a virus, as most people were prone to move data around via this medium. Today the Internet is the easiest way of transferring data, including viruses. There is a pit fall for a potential author of an email virus, emails leave an audit trail; and no matter how well one tries to cover this trail it is doubtful that it can be completely hidden, especially from organisations with almost unlimited detective resources. P2P systems could prove to be a step nearer anonymity for viral authors.

### **7.3. Accidents, exploits and attacks**

These are the actual damaging events that can occur.

#### **7.3.1 DoS - denial of service**

DoS attacks are used to make a service unavailable, they attempt to overload the host of the service with connection attempts that are entirely bogus. Each connection attempt utilises available ports, making them unavailable for genuine use, furthermore each connection attempt incurs a processing overhead on the part of the server, obviously too many of these can lead to an overloaded server and an eventual crash.

#### **7.3.2 DDoS - Distributed Denial of Service**

This more complicated form of attack involves the use of several of the attacks discussed so far. A typical scenario is dropping agent software on users machines by using a Trojan horse program; this agent can then just sit there awaiting instructions. Once this has been achieved on a number of machines the perpetrator of these intrusions can send requests to each of the agents to launch a DoS attack on a specific server. The result of all this is a more successful DoS attack that is very hard to trace back to the original perpetrator.

#### **7.3.3 Email spamming and bombing**

Unsolicited commercial email has become a problem in recent years, not only is it unwanted by most recipients but it uses up valuable bandwidth in networks. There is a potential for P2P systems to be used as mass market Spam engines, even if they are not intended for such a purpose. Email bombing is the act of sending an email message to a recipient in huge quantities, this is easily detected, usually by the perpetrators or recipients network service provider. Because of this, perpetrators are frequently looking for ways to remain undetected, and P2P systems have the potential to mask them.

### **7.4. General security solution areas**

Once the assets that need to be protected have been positively identified, and problems arising from contradictory ones have been tackled, the process of designing ways to protect these assets can begin. This will involve a close scrutiny of the proposed system architecture and an iterative method of design that pulls the system architecture, security requirements and security design closer together. An iterative method is needed here because the security requirements of a P2P system and its architectural design are highly liable to affect each other, so they must be allowed to evolve together.

To come up with a viable security solution, one must firstly decide upon what form attacks (deliberate and accidental) to a system may be and then set about creating failure models. It must be pointed out that preparing for accidental attacks, security holes that are chanced upon, is hard to achieve, and it is always prudent in either case to have a regular backup (where appropriate), fallback systems or redundancy or a combination of these. Even here problems arise, for example, a fallback system will kick in when a main system goes down, so at least a level of operation can be maintained. Fallback systems are not generally engineered as well to cope with attacks, and a potential attacker knowing this could well incur a service denial attack on the main system and wait for the fallback system to take over before launching an attack. Problems with a regular back up procedure can happen when undetected errors in data propagate their way through the system and are only noticed at a much later date, when a rollback is probably too late.

Cryptographic transport of data and authentication can help overcome many of the security vulnerabilities mentioned so far. It is pointless sniffing packets from a communication channel if they are encrypted and IP spoofing is equally as hard when authentication challenges can be issued by a server, clients authenticated, and an encrypted communication session begin, all without any vital security information being sent from either party (i.e. zero knowledge proofs).

P2P adds new problems to security, the mechanisms that allow conventional client-server authentication and encryption, may not be supported by the infrastructure used by a P2P system. Also the fact that the P2P system is such a fluid environment with nodes appearing and disappearing, each time potentially altering the topology of the system.

## 8. Appendix B – Miscellaneous issues with P2P architectures

### 8.1. Circumventing Firewalls and NAT

Firewalls can interfere with a peer-to-peer application's communication by blocking communication between nodes. A possible solution is the development of *smarter* firewalls that can intelligently interact with p2p applications. Another potential solution is to allow the ports used by a peer-to-peer application to be altered depending upon what ports the firewall will allow.

Data transfers across a peer-to-peer network could also use well-known ports, ones that nearly all firewalls will allow anyway (such as port 80 – HTTP); data could then be encapsulated within meta-languages such as XML. A possible *smarter* firewall could identify such XML communication and consequently allow use of the port (a form of data firewalling).

Although firewalls and NAT will not have a specific effect on the network architecture used they are something that can have an effect on the perceived dependability of a system (for example if a user can not communicate with another user) and thus need to be considered and countered.