

# Dependability Properties of P2P Architectures

James Walkerdine, Lee Melville, Ian Sommerville  
*Computing Department, Lancaster University, Lancaster, UK*  
{l.melville, walkerdi, is} @comp.lancs.ac.uk

## ABSTRACT

There is an increasing interest in using P2P as a basis for software systems. However, by their very nature, achieving dependability within a P2P system can be difficult. This paper attempts to identify the properties that can influence the dependability of Peer-to-Peer (P2P) architectures. In turn, this can then be used to help inform in the creation of more dependable P2P software.

## 1. Introduction

A system's dependability can be thought of as being the trustworthiness of the system. The difficulty when attempting to measure dependability is that it is typically a context sensitive property. While 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 other architectural properties. These properties include security, reliability, availability and performance.

Because of this influence it is difficult to consider dependability without also considering these additional properties. Furthermore, these properties can in turn be affected by other sets of properties. Consequently, a system's dependability can often be influenced by factors that would not typically be regarded as dependability properties in their own right. For example, a P2P system's perceived dependability can be influenced by its responsiveness, which in turn can be influenced by the node discovery mechanism used.

Because of this network of property influences within a P2P system, it is necessary to identify and consider all properties that can exist within P2P architectures, and to relate these to the dependability of the system.

Identifying dependability properties and achieving dependability within P2P systems is further complicated by the fact that numerous P2P network architectures exist and no single architecture is likely to be suitable for all application types. For example, applications such as Napster [1] ultimately benefit most from a semi-centralised

architecture, where as FreeNet [2] is most suitable to be run over a decentralised architecture.

The type of architecture used can influence the dependability properties of the system. Take for example, security. Fully decentralised P2P systems are likely to be better suited at handling denial of service attacks, semi-centralised P2P systems would be better suited for handling peer certification.

This paper aims to identify the main dependability properties (and related properties) that can play a part within P2P systems. This, in turn, can then be used to help inform the creation of more dependable P2P systems.

Because the choice of network architecture can have an effect on dependability properties, this paper will first provide an overview of the main P2P architectures, before going on to reviewing the different properties. The paper provides an initial analysis of the different architectures effect on these properties. Future work will provide a more detailed analysis.

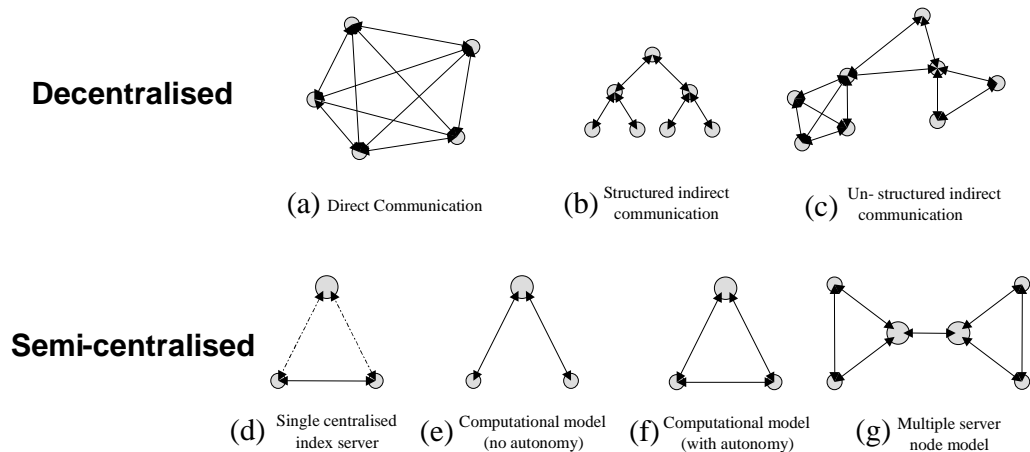
## 2. Overview of P2P Architectures

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 server node to a client node.

From examining existing peer-to-peer systems it is apparent that two core types of architecture exist. *Decentralised*, where each node within the network is regarded as an equal and no control nodes exists, and *Semi-centralised*, where there exists at least one control node that performs an authoritative role within the network. Figure 1 illustrates seven possible peer-to-peer network architectures. Their key characteristics are detailed below.

### 2.1 Decentralised Architectures

A decentralised peer-to-peer architecture is one that does not possess any central point of control or focus. Each



**Figure 1 - P2P Architectures**

node within the network is typically regarded as being autonomous and of equal standing. Figure 1 contains three decentralised network models.

#### *Direct Communication (a)*

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. It is feasible, therefore, for this architecture to be used for systems in environments such as small companies.

#### *Structured Indirect Communication (b)*

What differentiates this architecture from the direct communication architecture is that it is not necessarily the case that all nodes can communicate directly with one another. Instead, nodes that are not directly linked can communicate by sending messages via other nodes. Such an architecture can make use of numerous types of network topology, including hierarchical, 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). Consequently a degree of management may be required to ensure that the architecture’s structure persists.

A system that uses a structured indirect communication architecture is the Domain Name System (DNS) [3]. What is notable about DNS is that it still functions despite being scaled to 10,000 times its original size.

#### *Unstructured Indirect Communication (c)*

Unstructured indirect communication architectures are essentially the same as their structured counterparts with the difference being their more freeform nature. Nodes can be connected and removed from any part of the network, resulting in a varied density of nodes throughout.

The unstructured nature of this architecture removes the need for any management, however it does result in more

emphasis being placed on a discovery service, given that changes to the network can happen without all nodes being aware of them.

An example P2P system that adopts an unstructured indirect communication architecture is FreeNet [2].

## 2.2 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 strict control over the whole network, to simply acting as a central reference point for the remaining nodes.

Figure 1 contains four semi-centralised network models.

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

In this type of architecture there exists a single node that can act as a lookup for all other nodes within the network. This node could, for example, be used as a point of reference for the data or processing that is available on the network. All other nodes within the network are typically regarded as being equal and autonomous like in decentralised architectures.

Nodes within these types of architectures typically communicate directly with each other, however this is normally achieved with the aid of the central index node. A typical scenario could be where one node requests the location of a node they wish to contact from the central index node. Once a response is given, the node can then establish a direct connection with the target node.

By using a central index node the need for a discovery service is reduced (or even removed), and establishing a communication link between nodes can be quick and simple. 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 point of reference for the network. To cater for such circumstances a combination of a lookup index node and a discovery service can be used.

An example system that utilises such an architecture is the music file sharing system, Napster [1].

### *Computational without Autonomy (e)*

The two computational architectures are similar to the single centralised index server discussed above; a single node within the network acts as a focal point for 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 could control and manage the distribution of data and assign processing tasks to nodes within the network. Communication is also limited to between leaf nodes and the central index node.

Arguably, this type of architecture should not be considered as true P2P, given that the nodes within the network do not possess their own autonomy and are essentially slaves to the central node.

An example system that utilises such an architecture is SETI@home [4].

### *Computational with Autonomy (f)*

The alternative computational architecture is where the leaf nodes maintain a degree of autonomy. This autonomy could allow a node to operate intelligently making requests to the server rather than simply being dictated to, or allow the nodes to communicate with each other on the network. Again, for the latter, it is likely that the central node would facilitate any communication between nodes.

This architecture is essentially the same as the single centralised index architecture (d) except it is geared more towards distributed computation.

### *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 increasing a P2P systems reliability by removing a potential single point of failure, or improving Quality-of-Service should a single server become overused.

A similar approach to that used within single centralised architectures can be used for node communication, the only difference being 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 to connect leaf nodes 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 more global semi-centralised architecture.

## **3 Properties of P2P Architectures**

The previous section provided an overview of the key network architectures that can be used as basis for P2P development. This section moves on to identify the main properties that can exist within P2P systems that use these architectures. The properties have been split them into two categories.

*Architectural properties* – properties that can be specifically affected by the type of architecture

*Emergent properties* – properties that emerge over time as the architecture is used.

### **3.1 Architectural properties**

#### *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. 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 contain faults yet be seen as reliable by users of the system.

Reliability is important within a P2P architecture due to its distributed nature. A P2P architecture is particularly prone to network failures and so mechanisms need to be considered that can reduce the impact on the system should one occur. Semi-centralised architectures, in particular, suffer from possessing single points of failure.

#### *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 1000 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 these does not represent the number of clients in use at one time [6]).

Accordingly, catering for scalability should play a significant role when deciding on the architecture of a peer-to-peer system.

Direct communication architectures will not scale well. The rest however are more adaptable, although quality-of-service may suffer.

## *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. 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 system's ability to ensure that only information that is required is revealed to those people who need to use it.

Ensuring security within P2P systems is more difficult than with standard centralised server systems. On the one hand there are issues such as authenticating the people you communicate with and sharing information only with the people you trust. 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.

It is possible to classify security risks into three categories, causes, hazards and attacks. 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. For example, weak authentication mechanisms could lead to the system being accessed by unauthorized individuals, which could lead to the disclosure of sensitive information to unauthorised people.

Because a P2P system can be viewed as a synergistic system, the emergent system is composed of the individual nodes that are connected to it. This can create security problems, as it does in any component-based system that incorporates message-passing communication. 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.

Generally centralised nodes can aid security implementations but become obvious points of attack.

## *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 property.

*Adaptation and evolution to reduce effectiveness of future attacks.* The system must improve its survivability based on any 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.

## *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. 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 can have an important influence on the P2P architecture used. Generally, better degrees of safety control can be achieved with semi-centralised architectures, due to the central points of control.

## *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 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. 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 must 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.

Due to central points of control, more centralised systems would be easier to maintain.

### *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). A system that fails in this department cannot only irritate its end users but also can have a dramatic effect on its commercial uptake.

Some architecture's may be more attributed to certain types of application than others and mirroring this is the affect of applications on the network infrastructure it is built upon.

Generally users of a system will require a fast and consistent response to interactions with their local applications. Direct communication architectures would provide the best responsiveness, due to not requiring a discovery mechanism.

### *Responsibility, accountability and reputation*

A key challenge within P2P systems is enforcing rules of social responsibility. 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]. 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.

Micropayments 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.

### *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 can be more problematic, especially if the nodes collaborate.

Generally, it would be easier to monitor the status of a system with a more centralised architecture, due to the central points of control.

### *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.

As with safety, fault tolerance is easier to achieve with centralised architectures due to the more central points of control.

### *Political and legal independence*

In some circumstances it might be the case that political or legal actions can 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 resistibility. 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.

#### *Data integrity*

It is important for the data that is stored and manipulated by a system to maintain its integrity. 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 incorrect business decisions being made.

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 most difficult 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.

#### *Connection bandwidth*

How 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 T2 connection at a work. 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.

#### *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. Indirect communication architectures can be particularly affected due to the reliance on third party nodes for routing.

#### *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 a semi-centralised architecture 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.

#### *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.

In terms of network architecture, a semi-centralised model can 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. 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. For decentralised architectures, user information could be withheld entirely from the rest of the network, but more specific network information (such as IP addresses) would likely have to be made available.

#### *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.

#### *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. More centralised architectures have the advantage in that they are more suitable for network monitoring.

#### *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. Based

on this, semi-centralised architectures would be best suited for achieving manageability.

#### *Adaptability*

P2P networks are very unpredictable in nature; machines with different specifications can be connected to and from them at will. A P2P architecture needs to be able to adapt when such changes occur to ensure its continued operation. Adaptability is an architectures 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. Network evolution is discussed in section 3.2.

How successful a P2P network is at adapting itself can be influenced by the type of underlying architecture used. Indirect communication architectures are the most adaptable, however semi-centralised architectures provide a more manageable alternative.

### **3.2 Emergent properties**

#### *Person centric addressing*

Traditionally, individual machines have possessed an IP address that uniquely identifies 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 as highlighted in a previous section, dynamic IP's pose their own problems for peer-to-peer applications. Another technical solution is with the introduction of IPv6 [11], 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 shifted the focus away from machine centric addressing to *person centric addressing*. Rather than assigning a unique ID address to each machine, an ID address 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.

Such a change can be compared to 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 owner. Existing peer-to-peer applications act in a similar fashion. An address is mapped to the user, not to the machine, no matter which machine that user is using.

#### *Network evolution*

Studies of existing P2P systems have shown that the network architecture used can evolve over time [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 [6] it was observed that the nodes with more resources would move to the centre of the 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.

#### *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.

#### *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 (CA's), entities that can vouch for a certain user/device. CA's 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.

## **4. Summary and Conclusions**

This paper has attempted to identify the properties that can have an influence on a P2P systems dependability.

Dependability is a difficult attribute to measure. Not only is it context sensitive but a range of interconnected properties can also influence it.

Achieving dependability within P2P systems is further complicated by the fact that numerous underlying network architectures can be used, and these are likely to have an impact on the dependability properties.

This paper has provided an overview of the key P2P network architectures that are used, before going on to identify properties that can influence a P2P system's dependability.

Currently only an initial analysis has been performed in determining the affects the different architectures can have on the dependability properties. This has shown that the different architectures can provide advantages and disadvantages. Direct communication architectures might be the most responsive, but do not scale well. Semi-centralised allow for a more managed system, but suffer from single points of failure.

It is our intention to extend the work by assessing any possible affects in more detail. The results of this work will be presented in a future paper.

## **5. References**

- [1] Napster. More info at the URL <http://www.napster.com>
- [2] FreeNet. More info at the URL <http://freenet.sourceforge.net>
- [3] Domain Name Service (DNS). More info at the URL <http://www.rad.com/networks/1995/dns/dns.htm#doc1>
- [4] [SETI@home](mailto:SETI@home). More info 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 info at the URL <http://www.gnutella.com>
- [10] ICQ. More info at the URL <http://www.icq.com>
- [11] IPv6. More info can be found at the URL <http://playground.sun.com/pub/ipng/html/#IPNGWG>