

PEER TO PEER VS CLIENT-SERVER : WHICH IS BETTER?

PETER KEOGH
DT228C
ADVANCED SOFTWARE DEVELOPMENT
D94140029

1. INTRODUCTION

Ubiquitous Computing can be defined as computation, communication and sensing being integrated with the physical world. First described in the paper by Mark Weiser “The Computer in the 21st Century”, it is a paradigm where computers would “disappear. They (would) weave themselves into the fabric of everyday life until they are indistinguishable from it.”[1] The pervasiveness of computers is the defining characteristic of Ubicomp, as a result it’s applications touch on and integrate a broad range of topics within computer science and electronic engineering. Ubicomp has the potential be utilised in potentially any industry or part of life.

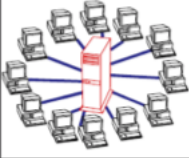
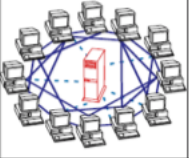
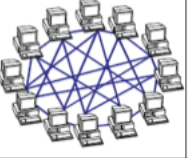

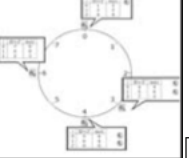
The choice of network architecture in the design of a ubiquitous computing system is crucial to the success of the system. The growing number of devices owned by a single user and has increased the amount of data produced and exchanged by systems. This has brought with it an increased load on networks exchanging this data. As ubicomp systems become more and more common this load is set to increase. Therefore determining how data will be exchanged and stored in a systems architecture is a critical factor for the success of ubiquitous computing system.

With such a broad range of devices and potential applications it is impossible to truly say that one network architecture is better than another. Client-Server networking and Peer-to-Peer networking both have their strengths and weaknesses. We cannot look at one and say that it is better than the other; we can only look at them and determine which is better for a particular application. This paper will present information that will hopefully help guide the decision making process when developing an application.

The popularity of these architectures can be easily seen when we look at their respective shares of network traffic. Fixed web and data, mostly client-server services currently account for 5,625 Petabytes of data transfer a month. Similarly P2P file transfer consumes 5,254 Petabytes of data worldwide per month.[2] In previous years the share would have been far greater; the decline in the use of bit torrent clients and the growth in both mobile web and video over IP have changed the landscape somewhat.

P2P and Client-Server are not restricted to being independent of one another. In fact, companies such as Spotify and Skype use a hybrid approach. The advantages of hybrid architectures should be evident in the description of the architectures later. The view taken in this paper is that neither is better than the other; the suitability of the architecture depends on the application. Many ubicomp applications will benefit from a mix of both architectures.

2. THE ARCHITECTURES

Client-Server	Peer-to-Peer			
<ol style="list-style-type: none"> 1. Server is the central entity and only provider of service and content. → Network managed by the Server 2. Server as the higher performance system. 3. Clients as the lower performance system <p>Example: WWW</p>	<ol style="list-style-type: none"> 1. Resources are shared between the peers 2. Resources can be accessed directly from other peers 3. Peer is provider and requestor (Servent concept) 			
	Unstructured P2P			Structured P2P
	1st Generation		2nd Generation	
	Centralized P2P	Pure P2P	Hybrid P2P	DHT-Based
	<ol style="list-style-type: none"> 1. All features of Peer-to-Peer included 2. Central entity is necessary to provide the service 3. Central entity is some kind of index/group database <p>Example: Napster</p>	<ol style="list-style-type: none"> 1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities <p>Examples: Gnutella 0.4, Freenet</p>	<ol style="list-style-type: none"> 1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → dynamic central entities <p>Example: Gnutella 0.6, JXTA</p>	<ol style="list-style-type: none"> 1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities 4. Connections in the overlay are "fixed" <p>Examples: Chord, CAN</p>
				

[8]

2.1. Client-Server.

Many successful technology businesses are based on the Client-Server model, most obviously in Cloud Computing. Gmail, Facebook and Dropbox are all examples of applications built on the client-server architecture. It is an appealing architecture for these businesses as they can store and access customer information easily. The advent of faster and more reliable internet access in many parts of the world and the growth in the number of connected devices owned by a single user has also contributed to the growing popularity of this architecture.

Users want to be able to access their resources on all of their devices. Client-server architectures make this possible. In years gone by users would have to back all their work up on a CD or floppy disk. They would forget to do so at their peril. Now a user can work on a project locally and easily save it to a remote server to back it up.

Using cloud services is appealing to users as it can allow the user to spend less money on their devices. "Thin" clients like Google's Chrome book have come to fill a new gap in the market. As services such as Google Docs provide a cloud alternative to Microsoft Word the processing and storage requirements of a laptop can be minimised.

In the Client-Server Architecture a server services requests from clients. The server is always on in order to service the client's requests. The client passes data to the server which the server can store or perform some computation with, returning the result to the client.

The client server architecture is typically used in web applications. In a web application a client, usually a web browser, sends a request to a server using the http protocol. The server then responds by sending a html page for the browser to render. In a client-server application clients do not directly communicate with each other. They can communicate with each other via the server. For example, in the Facebook chat application a client

can send a message to the server using AJAX which the server forwards to another client. To the user it appears the same as if the two clients were communicating directly. The server has a fixed IP address which is known or can be discovered by any client that wishes to connect to it.

Increasing numbers of clients typically put more strain on a server. As one host needs to accommodate many requests it requires a robust infrastructure. Usually a single host isn't enough and more complex distributed application architectures need to be developed. These architectures typically make use of server farms; large warehouses filled with servers that are capable of communicating.

2.2. Peer-to-Peer.

P2P first emerged in the popular consciousness in the form of Napster. It has caused quite a stir in the past and continues to do so. It's most common use is file sharing, often illegally. It's use in this manner has had a major impact on the entertainment industry and as a result they have lobbied continuously for it's regulation.

P2P offers advantages over the client server model. In the case of file sharing P2P provides a greater availability of popular material. The more popular a resource is the more likely a peer is to be sharing it. As a result it can be more robust than a client server model, in which, once the server goes down the client is unable to avail of it's resources. On the other hand, if a resource is unpopular it is unlikely that many peers will be seeding that resource at any point in time. In this case a client server architecture might be preferable.

In the Peer-to-Peer devices communicate directly and intermittently. The devices in a P2P architecture are known as peers. P2P application that require a server are considered less "pure" than those who don't. An application with no reliance on servers could be said to be "pure" P2P. P2P provides many interesting development opportunities; for example, in applications that require collaboration, decentralisation or high network throughput. This has lead to it's use in popular applications such as BitTorrent and BitCoin.

A P2P architecture is 'self-scalable'. While the addition of an extra peer increases the workload of the system by increasing the number of requests the additional peer also increases the ability to handle that workload by providing the service to other peers. Securing P2P applications can be difficult due to their open and distributed nature.

P2P can be broken down further into four subcategories of architecture; centralised, decentralised, hybrid P2P and DHT based.

An example of centralised P2P would be BitTorrent. In order to communicate with peers in the network a client must obtain information about the peers from a centralised server known as a tracker. Information about a tracker is obtained from a metadata file. A user needs to visit centralised server such as The Pirate Bay or Kick Ass Torrents in order to obtain this file. This dependence on a centralised server to track the network is what defines BitTorrent as centralised P2P.

In a decentralised or pure P2P Architecture each peer acts as an indexed server. It also acts like a router by relaying queries between peers. An example of this would be BitCoin or Gnutella 0.4.

In hybrid P2P hosts with a good internet connection act as routers for other peers in the network who's connection isn't as strong. An example of this is Gnutella 0.6.

Peer-to-peer systems are most effective at storing large amounts of immutable data. They perform less well at storing mutable data.

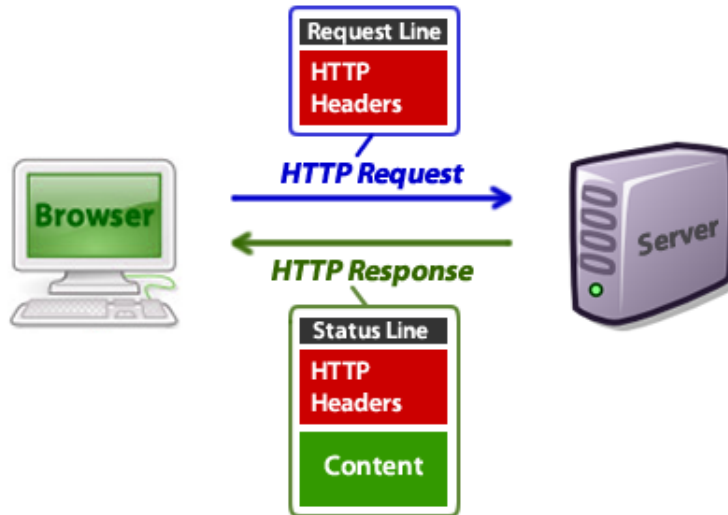
3. EXAMPLE PROTOCOLS

A protocol is an agreed method of communicating between applications. The protocols outlined below provide details of how current technologies using the architectures outlined above communicate. While these particular protocols may not be directly applicable to Ubicomp directly they provide a useful outline of how the architectures need to work in reality. Designers of Ubicomp systems are likely to encounter similar problems that system designers have encountered in the past. These protocols outline methods for overcoming these problems. An outline of these protocols will be useful in reasoning about the architectures as applied to Ubicomp systems later.

3.1. HTTP Protocol.

The client-server architecture is the typical architecture used in web applications. The protocol used for transferring information between the client and server is the HTTP protocol. HTTP (HyperText Transfer Protocol) follows a request response cycle. The client makes requests to the server and the server responds to the request. The server doesn't send any information to the client if not prompted by a request. This has implications for applications using the protocol.

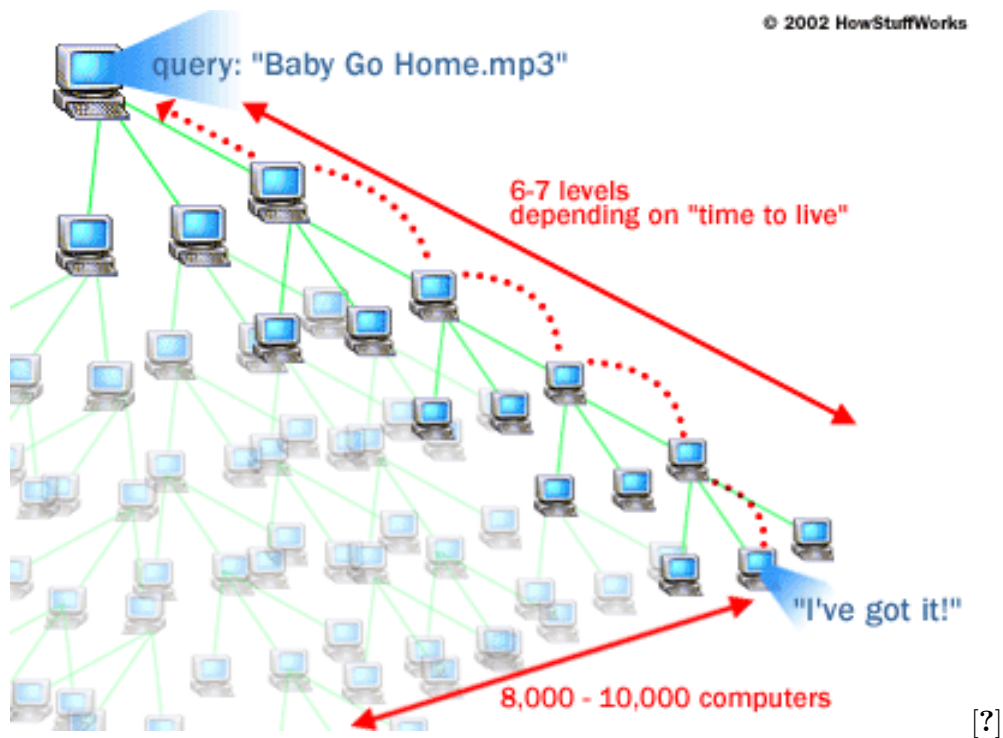
If nodes wish to collaborate with each other or obtain realtime information about other nodes in the system they must "poll" the server by repeatedly sending requests to the sever. This is very inefficient as it results in significant network overhead and results in the server processing requests even when the state of the system remains unchanged. A solution for this problem has recently emerged the the standard of web sockets which allows servers and web browsers to set up a dedicated end to end bidirectional connection.



[?]

3.2. Gnutella 0.4. [5]

Gnutella was the first large scale decentralised P2P network of its kind. It “builds at the application layer a virtual routing mechanism.” [5] Early versions of Gnutella were truly decentralised P2P networks. No centralised servers were needed to facilitate the transfer of files from node to node. One of the implications of not using centralised servers in a P2P application is the challenge of keeping track of network nodes and what resources are available in the network. In order to obtain this information “network flooding” was used. Peers all maintain at least one TCP connection to other nodes in the system. The network flooding algorithm “floods” the network with requests; each node sends messages on to all of its neighbours except the one it received the message from.



3.3. Gnutella 0.6. [5]

The flooding model used in Gnutella 0.4 had scalability problems. Searches were costly and usually terminated prematurely. Gnutella moved from a “pure” P2P paradigm to a hybrid approach. This new approach makes use of ultra peers and leaf nodes to create a hierarchically structured network. An ultra peer is a node that has a faster connection and elects to route more traffic for the network. The ultra peers also act as proxies for the leaf nodes. Searches become more efficient since searches only need to flood the ultra nodes instead of the whole network.

3.4. Bittorrent Protocol. [6] [7]

The bit torrent protocol is a P2P file sharing protocol that runs over HTTP and TCP. The owner of a file hosts a “bencoded” metadata file (with the extension .torrent) on a server. A user downloads the metadata file and opens it in a bit torrent client. This client uses a url contained in the metadata file to connect to a “tracker”. A tracker is a HTTP service that maintains information about the torrent, including a list of peers. The tracker may also provide stats for the torrent at a “scrape” url.

Once a client has obtained information from the tracker it can communicate with peers using TCP. Peers exchange the blocks defined in the metainfo file. Peers all have connections with each other that they maintain state info about. Peers exchange information based on these states.

These states are choked and interested. A peer may choke a client to stop it requesting data. A peer is interested if the client has something that the peer wants. If a peer is interested it can unchoke the client so they can begin exchanging data.

4. COMPARISON OF PERFORMANCE

Systems that are built with the Client-Server architecture and P2P systems are both examples of distributed systems. There are several challenges associated with distributed systems that provide interesting metrics to analyse these architectures under. These are:

- Performance, the amount of work a system can do.
- Scalability, the ability of the system to handle greater loads and to get bigger.
- Security, the ability to keep malicious parties from doing things to or with our system we don't want them to do.

4.1. Availability, Fault Tolerance and Scalability.

Availability in client-server systems is dependent on the server architecture and network connection. If a firewall prevents a user from accessing a server then to that user the server is unavailable.

A single server may be enough to handle requests for a service with few customers making infrequent requests. In the case of a server simply providing static information for clients, a single server can usually handle all the requests that are made of it. However, more popular and complicated services that require more processing and data throughput need more complicated server architectures.

A service that works internationally can reduce the network latency of requests in certain regions by having the same service running on multiple servers in locations that are closer to their customers. This reduces the load on a single server and has the added advantage that we can partition customer's data to be located closer to them.

Data can be managed through replication and partitioning. Data can be partitioned so that data that are frequently accessed together can be stored together. By replicating data the system becomes more fault tolerant; if one server crashes the data can be accessed at another.

All of the techniques for improving availability, fault tolerance and scalability in client-server architectures come at a cost. There's the financial cost of paying for more storage and processing power. There's also cost associated with developer time. Developing and deploying distributed server architectures can be time consuming and expensive not to mention complicated. While replicating data improves the fault tolerance of the system the system now needs to make sure that the data is consistent throughout the system. There is also an additional cost associated with system administration.[8]

Availability in P2P systems is a function of the number of system nodes and the popularity of the resource to be accessed. If a node wishes to request a resource that is popular in a network with a large number of nodes then it should have no problem getting it. It is likely that there will be nodes geographically nearby that are hosting the information and are currently available for processing. This would result in reduced network latency and a faster download of the resource.

One problem in P2P networks is the availability of unpopular resources. Niche resources like academic journals are better hosted on a server. On the other hand popular resources are highly available; their availability increases in proportion to their popularity.

P2P is also extremely fault tolerant if the network is big enough. If a cup of coffee spills over one of the nodes in a large network the downloading peers are unlikely to

notice a performance difference. P2P is at it's most fault tolerant when it is completely decentralised and has no reliance on centralised trackers.

P2P networks are highly scalable as a result of all of the above since an increase in the number of clients consuming network resources increases the ability of the network to handle it's load.

4.2. Security.

All software systems that are networked have similar vulnerabilities. The network connection can be eavesdropped on. Data stored can be accessed without permission. It is considered a best practice to encrypt any sensitive data that is transferred or stored.

One of the most common attacks on a server is a denial of service attack. An attacker floods a resource with requests to make the resource unavailable for others.[8] One of the advantages of Client Server architectures is that once a security risk is identified it can be fixed for the whole system by a system administrator. This is in contrast to P2P which can't be attacked by a denial of service but where if there is a problem software must be updated on a huge number of machines by a huge number of owners.

One of the main methods of attacking a peer-to-peer network is to have multiple nodes conspiring against the other nodes in the system. In the case of bitcoin the only way to achieve this is to have more than half of the nodes behaving maliciously.[?] In order to make this happen the attacker would need access to a number of hosts beyond the budget of any hacker. One strategy for attacking a P2P network using conspiring nodes is to provide false information to other nodes in the network.

An obvious security issue in P2P systems is the P2P application code has access to the filesystem of a user. In the case of systems that share files like BitTorrent users need to know that they can trust the code of the torrent client they use. In a more complicated P2P system where users could contribute arbitrary code to be run there would need to be mechanisms in place to protect the system from this code.

5. SOCIAL AND POLITICAL ISSUES

There are a number of important political and social issues surrounding the choice of architecture for a particular application.

5.1. Jurisdiction.

The client server model has brought with it legislative challenges for governments. European data protection authorities have had trouble enforcing local legislation with companies based abroad. Multinational companies such as Google tend to argue that since they are based in California then EU Data Protection laws don't apply to them. Others argue that the physical location of data determines which jurisdiction it resides in.

Even if this is the case, companies may have to obey the laws of their home country if they control the data regardless of the location of the data. Data stored on European servers by an American company may be given to the United States Government under the Patriot Act. Both Amazon and Microsoft have said they would give the US Government access to their European cloud under the Patriot Act.[?]

5.2. Censorship.

The client server model used in traditional web applications is far easier for governments to censor than peers to peer networks. Governments can force internet service providers to block IP addresses, redirect domain name requests to bad IP addresses or force a hosting company to remove a service from it's servers. This is the case in countries like China where the government's "Great Firewall" prevents Chinese netizens from accessing sites like Facebook, Google and Twitter. These services have come to be replaced by Chinese equivalents like Baidu, Renren and Sina Weibo. These sites are subject to heavy regulation from the Chinese government.

The centralised nature of the client server model makes it easy for governments to censor applications built on it. One of the reasons for this is because it is easy to shutdown a system with a single point of failure.

One of the methods of bypassing such censorship is by leveraging the P2P model. In January 2014 The Pirate Bay announced that it was developing "a browser-like client to circumvent censorship, including domain blocking, domain confiscation, IP-blocking." The software works like both a web browser and a BitTorrent client. Web application files that would usually be stored on a centralised server are instead shared among peers. The application has it's own DNS system in which it uses a fake DNS that links to a unique and verified public key.[11]

Another example of a project that utilises the P2P model to bypass web censorship is the Freeweb project conceived by Shen, Liu and Zhao.[12] This model functions differently to the Pirate Bay system. In Freeweb nodes in regions not subject to censorship act as proxies for nodes in regions that are.

5.3. Privacy, Spying and Hacking.

Services such as Facebook and Google offer ways of communicating and discovering information that they collect in their servers. While most people consider these services free they do come at a price. It's an internet cliché now that "If You're Not Paying for It; You're the Product." In light of the revelations revealed by Wikileaks and whistleblowers like Edward Snowden the public have started asking if the convenience of these services is worth trading their privacy for.

Client server technologies offer us the opportunity to automatically back up data to a cloud account. Services like Apple's iCloud allow users to synch their iPhone data automatically. Many users simply set this up and don't think about it again until something goes wrong with their device. Users may be unaware of what they are synching. Having so much personal data automatically synched to a cloud service provides a gold mine for hackers. Recently many hackers have released personal photos of celebrities such as Jennifer Lawrence that were synched to iCloud.

5.4. Copyright and Free Riding.

P2P networks such as bit torrent have been notoriously used for illegal file sharing since Napster. Attempts were made to crack down on this file sharing in legislation such as the Stop Online Piracy Act (SOPA). SOPA was eventually abandoned after websites such as Reddit and Wikipedia protested the legislation with a blackout day. The size of P2P networks and the amount of file sharing that happens within them makes it extremely difficult and expensive to prosecute.

One of the issues in P2P networks is the social dilemma of “free riding”. In a file sharing system the availability of a resource is dependent on peers sharing that resource. However, if peers choose not to share the resource they can improve their network bandwidth and download more resources for themselves. A consequence of this is that as fewer and fewer peers are sharing it becomes easier to prosecute file sharers. [13]

6. NOTABLE SYSTEMS AND THE ARCHITECTURES IN THE CONTEXT OF UBIQUITOUS COMPUTING

It’s easy to compare P2P and the client-server model as they are being currently utilised. While it is tempting to believe these models to be analogous to these architectures in Ubiquitous computing, that is not necessarily the case. The requirements of a Ubiquitous Computing system are different from systems that have been used in the past. For example, ubiquitous computing systems are likely to produce large amounts of data as a result of the numbers of sensors in the systems. Storing all of this data on server would probably be overkill not to mention expensive.

The choice of network architecture impacts the development of technology. A prime example of this is in the development of the World Wide Web. Web browsers are no longer simple programs that display HTML. One of the motivations for this has been to reduce the load on servers. Why should a server waste precious processing power assembling HTML views that can be created on the fly by a browser.

This can be contrasted with Wireless Sensor Networks. Currently sensors, particularly on the nano scale have little processing power or memory. It is possible to store other sensors information and network in a P2P fashion. However, these sensors aren’t cheap and there are issues with the technologies such as power consumption.

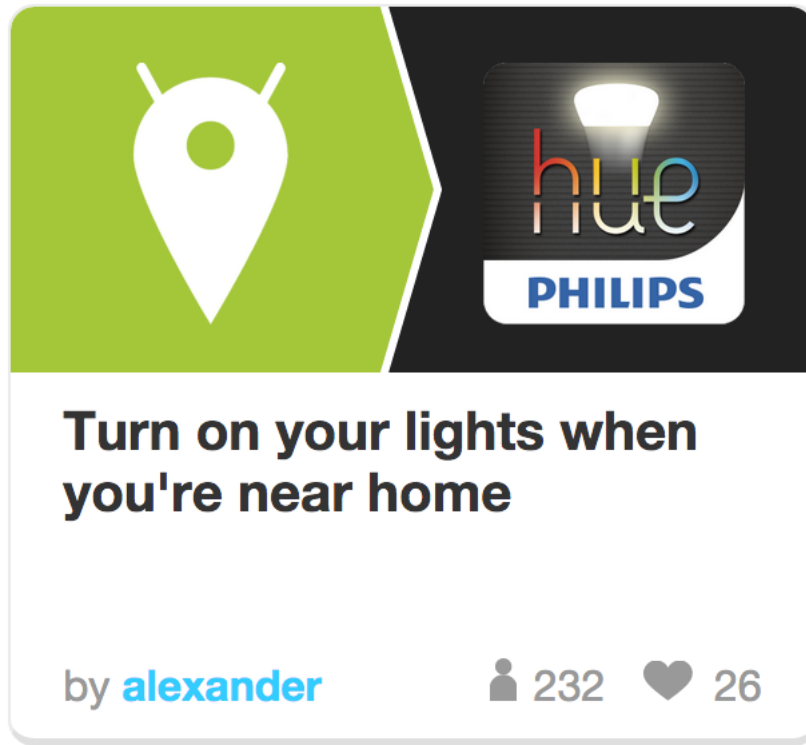
What follows is a collection of interesting technologies. Some of the technologies have been included as they illustrate the advantages of a particular network architecture over another. Others have been chosen because they have direct applications in UbiComp or are part of the emerging research in the area.

6.1. IFTTT. [?]

If This Then That (IFTTT) is a web application that allows users to link web services together in “recipes”. It allows people with no programming experience to construct conditional statements using powerful Web APIs. IFTTT allows a user to coordinate actions across many popular web applications such as Twitter, Dropbox, Evernote, LinkedIn etc. The user can also set actions based on conditions of their device. In the simplest sense a “recipe” can be defined as a conditional statement and a corresponding action. If a condition is true in one system then take an action defined in another. Users can share these recipes with other users.

Users can define an response based on the location of their android phone. This allows them to define the kind of context aware actions that would be typical in UbiComp. At the time of writing the eighth most popular recipe on the site is “Mute my Android device when I get to the office turn on vibrate”. Perhaps even more impressive is the recipe that takes advantage of Philips Hue to “Turn on your lights when you’re near home”.

This application nicely illustrates the type of ubiquitous computing applications that the client server model is suited for. Having one centralised computer (in this case a server) coordinating actions across multiple devices is a design pattern likely to arise in the development of Ubicomp systems. It is particularly suited to coordinating actions in one device many users situations; for example many users sharing one the same lights in a house.



6.2. BitCoin. [?]

BitCoin is a digital currency that utilises the P2P architecture to remove the need for centralised institutions in order to conduct financial transactions online. E-Commerce has traditionally depended upon a trusted third party institution to guarantee transactions between parties. Using this third party incurs an additional cost in the transaction. BitCoin attempts to remove the need for these centralised bodies by using a complex P2P algorithm.

Every node in the network maintains a copy of the public ledger of bit coin transactions known as the Block Chain. Nodes in the network “mine” bitcoins by computing cryptographic hashes in order to validate and record the transactions of other BitCoin users. It may take some time before BitCoin and other crypto-currencies are widely adopted. Currently its value is highly volatile so it will be difficult for mainstream users to trust.

BitCoin illustrates how functionality typically thought only possible in client server architectures can be produced in a P2P network by using advanced computing techniques.



6.3. Wireless Sensor Networks.

A Wireless Sensor Network is a network of wirelessly connected autonomous nodes with sensing capabilities. WSNs have a wide range of functionalities and applications. WSNs have been used in applications monitoring glaciers, oceans, vital signs and zebras.[14] Sensors that can wirelessly communicate information to other devices about their environment have applications useful for Ubicomp, particularly in the area of context awareness. It is important that these sensors can communicate amongst each other and with external devices effectively. As a result there is currently much research involving WSNs and P2P communication.

One example of such research is the TinyTorrents[15] application that integrates WSNs and the BitTorrent protocol. WSNs are typically required to be both small and low cost. As a result this puts further constraints on memory, networking capability, battery power and energy consumption. In the paper the authors outline advantages of applying the BitTorrent protocol to WSNs as follows:

- Security; The data is segmented into pieces that are replicated so tampering with a single node has less of an effect on the integrity of that data.
- Reliability; Each data segment is sent in a single packet so the probability of successful delivery is increased.
- Efficiency; BitTorrent employs the “rarest piece” algorithm resulting in efficient replication of data.

Another paper describes efficient P2P message communication in Wireless Sensor and Actor Networks (WSANs). In WSANs the sensors are static and actors move around collecting information from the sensors. Since there is poor communication between actors the paper presents methods for communication between the actors via the sensor network using P2P. The communication is made energy efficient by leveraging the fact actors typically visit parts of the network more than others.[16]

6.4. Workpad Architecture. In the paper “Pervasive Software Environments for Supporting Disaster Responses” the authors discuss a system that has been developed in Italy to aid first responders. The architecture for this application makes use of both a peer-to-peer communication strategy and a client-server strategy. When first responders respond to an incident they each have their own local devices that they use to communicate with each other. These devices work together in order to form a peer to peer

mobile ad-hoc network. This is essential in a situation such as an earthquake where vital resources for communication such as base stations may be damaged. A single node in the MANET acts as a bridge between the network and a backend system. The communication between a single MANET and its associated back end system happens in a client server manner via the bridge. In the Workpad architecture the backend system is a federated group of backend systems that communicate in a peer to peer manner. These backend systems are typically the control rooms of different organisations e.g. fire departments, coast guard, hospitals. These systems act as peers provide ontologies which describe the data they share. An important function of the MANET is Adaptive Process Management; managing MANET task processing in the face of a changing network. An example function of Adaptive Process would be when one node moves out of range of the system a nearby node is instructed to follow him until he is in range of the network again.

6.5. Cloudlets. It looks like the client server model may be redefined in the near future with the development of “Cloudlets”. One of the benefits of cloud computing is the ability for a device with limited computational power to send information to be processed “in the cloud”. Ubiquitous computing can certainly make use of this model. An example of this would be performing real time computer vision techniques on video captured from a mobile device. Most mobile devices would be incapable of performing the computation of such algorithms at an advanced level. Even if they were, the power consumption would deplete a large amount of the battery’s energy. A solution would be to pass this processing to a server in the cloud, however, this solution brings with it another problem; the latency and low data throughput of a wide-area network. The wide-area network causes glitches and delays that are intolerable for a user. This latency is also unlikely to improve as over a WAN the data must be encrypted and pass through firewalls. This is the problem that Cloudlets seek to solve.

Cloudlets are described as a “data center in a box”; like a cross between a Wifi hotspot and modern clouds. It is a server that provides computation for users on a LAN or WLAN. Cloudlets don’t persist data locally, although they may send data on to a user’s cloud account. According to researchers latency is improved since WLAN is typically two orders of magnitude faster than GSM. In order to implement cloudlets the researchers took a virtual machine based approach; each user has a virtual machine associated with them. One approach to implementing this would be to use VM migration where a VM is suspended and its state transferred to another cloudlet. Another approach would be to dynamically synthesise a VM after receiving a VM overlay from a mobile device.

Cloudlets would make a huge number of difficult and interesting applications available to mobile devices. This might include intense graphics processing for realistic augmented reality, decision support systems or real time language translation from audio.

6.6. Smart Objects.

6.7. JXTA Overlay.

7. CONCLUSIONS

The purpose of this paper was to compare the Client-Server and Peer-to-Peer architectures. By exploring the topic from several different points of view it seems obvious

that each architecture has both strengths and weaknesses. It is difficult to say that one is truly better than the other. Client-Server applications have many advantages from a business point of view as they are easier to develop, analyse and maintain. Peer-to-Peer applications can be incredibly robust and efficient if designed correctly. With the wide variety of applications that Ubiquitous Computing encompasses it is likely that both architectures have their uses. Some applications will benefit more from one than the other and some will benefit from a hybrid of both. It will be interesting to see in the coming years the challenges developing ubicomp applications will present and how system designers react to these challenges.

REFERENCES

- [1] Mark Weiser. The computer for the 21st century. *Parallel Processing (ICPP), 2013 42nd International Conference on*, October 2013.
- [2] Cisco. Cisco visual networking index: Forecast and methodology, 20132018. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html. Accessed: 15-10-2014.
- [3] TechSoup. Client server. http://www.ictknowledgebase.org.uk/uploads/RTEmagicC_clientserver.gif.gif. Accessed: 15-10-2014.
- [4] TechSoup. Peer to peer networks. <http://www.ictknowledgebase.org.uk/peertopeernetworks>. Accessed: 15-10-2014.
- [5] Matei Ripeanu. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 99 – 100, August 2001.
- [6] Bram Cohen. The bittorrent protocol specification. http://www.bittorrent.org/beps/bep_0003.html. Accessed: 15-10-2014.
- [7] Various. The bittorrent protocol specification. <https://wiki.theory.org/BitTorrentSpecification>. Accessed: 15-10-2014.
- [8] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. Distributed systems, concepts and design, fifth edition.
- [9] The Pirate Bay. Miley cyrus, bangerz. [http://thepiratebay.se/torrent/8996596/Miley_Cyrus_-_Bangerz_\(Deluxe_Version\)_2013-Album](http://thepiratebay.se/torrent/8996596/Miley_Cyrus_-_Bangerz_(Deluxe_Version)_2013-Album). Accessed: 15-10-2014.
- [10] The Pirate Bay. Everywhere, the dawning age of ubiquitous computing. http://thepiratebay.se/torrent/4506989/Everywhere_The_Dawning_Age_of_Ubiquitous_Computing. Accessed: 15-10-2014.
- [11] TorrentFreak.com. How the pirate bay plans to beat censorship for good. <https://torrentfreak.com/how-the-pirate-bay-plans-to-beat-censorship-for-good-140105/>. Accessed: 18-10-2014.
- [12] Haiying Shen, Alex X. Liu, and Lianyu Zhao1. Freeweb: P2p-assisted collaborative censorship-resistant web browsing. *Parallel Processing (ICPP), 2013 42nd International Conference on*, October 2013.
- [13] Daniel Hughes, Geoff Coulson, and James Walkerdine. Free riding on gnutella revisited: The bell tolls? *IEEE DISTRIBUTED SYSTEMS ONLINE*, June 2005.
- [14] KAY RMER and FRIEDEMANN MATTER. The design space of wireless sensor networks. *Wireless Communications, IEEE (Volume:11 , Issue: 6)*, December 2004.
- [15] Ciaran Mc Goldrick, Michael Clear, Ricardo Simon Carbajo, Karsten Fritsche, and Meriel Huggard. Tinytorrents - integrating peer-to-peer and wireless sensor networks. *Wireless On-Demand Network Systems and Services, 2009. WONS 2009. Sixth International Conference on*, February 2009.
- [16] Shibo He, Xu Li, Jiming Chen, Peng Cheng, Youxian Sun, and David Simplot-Ryl. Emd: Energy-efficient p2p message dissemination in delay-tolerant wireless sensor and actor networks. *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS/SUPPLEMENT, VOL. 31, NO. 9, SEPTEMBER 2013*.