UNIVERSITAT POLITÉCNICA DE CATALUNYA

Escola Técnica Superior d'Ingenyeria de Telecomunicacions

Master of Science Thesis

Design and Implementation of a Distributed System to evaluate Net Neutrality

Tutor:
Prof. Juan Carlos de Martin

Gianluigi Pignatari Ardila

June 2008

Resumen

Históricamente la Internet se ha basado en un servicio de "mejor esfuerzo", enviando los paquetes lo mas rápido posible, sin diferenciación de trafico y sin ninguna garantía en su funcionamiento. Sin embargo, desde la introducción del Deep Packet Inspection (DPI) los Proveedores de Servicios de Internet (ISP por sus siglas en ingles) están en la capacidad de identificar diferentes tipos de trafico en términos de servicio o aplicación, por ejemplo si el paquete pertenece a una conexión HTML o a una sesión BitTorrent. Con la evolución del DPI los proveedores están en la capacidad de identificar prácticamente cualquier tipo de tráfico pasando por la red y han argumentado que sus redes están sobrecargadas especialmente de trafico multimedia (voz, video, imágenes). En consecuencia ISPs han propuesto cambiar el paradigma de funcionamiento actual de la Internet a un sistema por capas, teniendo diferentes capas de servicio. Dichas capas estarían basadas en parámetros de calidad de servicio, como retardo y ancho de banda por aplicación. Además del cambio propuesto por los ISPs, algunos han decido implementar sistemas de DPI para caracterizar tráfico y comenzar a hacer pruebas de diferenciación de trafico filtrando data de sus clientes. Lo más preocupante en este aspecto es la privacidad, ya que los dispositivos DPI acceden a capas superiores a la capa IP y además ello implica el cambio del funcionamiento del Internet como se conoce hoy en día.

Para hacer frente a esta situación desde el punto de vista técnico, y tener datos y bases técnicas para evaluar objetivamente las prestaciones de los ISPs el Centro de Investigación de Internet y Sociedad NEXA, del Politécnico de Turin ha concebido el desarrollo de una aplicación distribuida y multiplataforma llamada Network Neutrality Bot (NEUBOT). El objetivo del NEUBOT es de medir cuantitativamente características de conexiones banda ancha, como ancho de banda, retardo, filtraje de puertos y/o aplicaciones; para determinar si hay algún tipo de degradación de servicio por parte de un ISP.

El proyecto consiste en el diseño de las especificaciones y el desarrollo de la primera versión del Neubot. La arquitectura básica del NEUBOT consiste en una aplicación cliente que será instalada por usuarios que voluntariamente quieran participar en el

proyecto. La aplicación es diseñada para ejecutar las medidas usando dos tipos de arquitectura de red. La arquitectura cliente/servidor se encarga básicamente de caracterizar la conexión de la aplicación cliente y diferentes parámetros de la conexión punto a punto. Por otra parte, se utiliza la arquitectura peer-to-peer para realizar medidas específicas sobre el funcionamiento de protocolos y servicios.

Para el diseño hay que tener en cuenta diversas consideraciones propias del entorno no controlado ya que la aplicación debe funcionar en Internet donde no se puede controlar ningún parámetro y segundo, se desconoce que usuarios participarán en el proyecto ejecutando el cliente. El diseño está compuesto por todas las especificaciones, medidas a realizar arquitectura del sistema empleando diagramas UML, las ventajas y desventajas de usar arquitectura cliente/servidor o peer-to-peer.

Posteriormente, se describe la implementación del Neubot. Primero se detallan aspectos de la arquitectura cliente/servidor, en esta sección Neubot maneja diferentes transacciones para realizar las medidas, una vez establecida la conexión con el servidor dicho canal (conexión TCP) es denominado el canal de control, luego se abre otra conexión TCP donde pasará la data correspondiente a las medidas. Las medidas realizadas son RTT, throughput TCP en download y upload (BTC) y throughput UDP en download y upload. Estas medidas son realizadas de manera secuencial entre el cliente y el servidor, una vez realizados los test el cliente se conecta con el servidor base de datos y envía todos los resultados de las medidas.

La arquitectura peer-to-peer es desarrollada con una tecnología llamada JXTA impulsada en un principio por Sun Microsystems, posteriormente por la comunidad de software libre. Esta plataforma permite desarrollar una aplicación peer-to-peer, ofreciendo las primitivas y herramientas básicas de comunicación necesarias en redes peer-to-peer, como por ejemplo sistema de nombre, protocolo de enrutamiento a nivel de aplicación entre los peers, pipes para la comunicación entre otros. Estas herramientas permiten desarrollar una serie de protocolos y servicios ligeros y flexibles. En el caso del Neubot, desarrollar la arquitectura con JXTA ofrece exactamente las herramientas necesarias ya que Neubot no representaría un típico sistema P2P de transferencia de archivos o llamadas o mensajería instantánea, en el cual se puede usar una implementación de algún protocolo P2P porque son muy rígidas y probablemente tendría un exceso de funcionalidades que no son necesarias para cumplir las especificaciones. Para cumplir el objetivo de realizar medidas de funcionamiento sobre protocolos específicos, JXTA es usado específicamente para diseñar un servicio de presencia para localizar otros peer a través de protocolos y primitivas de JXTA. El servicio de presencia permite al Neubot anunciar que está disponible para realizar medidas y si hay algún otro cliente disponible se conectan para establecer una sesión

del protocolo a medir. Una vez conectados los peers se activa el servicio de Neutralidad, este servicio se encarga de negociar los parámetros y el protocolo a medir; una vez establecido el protocolo se inicia una sesión y se intercambian datos. Para la primera versiń del Neubot que es la desarrollada en este proyecto se implementó el protocolo BitTorrent para realizar los tests, por ende se realiza una transferencia de un archivo entre los dos peers y se obtiene en primer lugar que la transferencia fue exitosa, luego la cantidad de bytes intercambiados y el tiempo de la descarga.

Mientras se realiza el intercambio de datos se utiliza una técnica para detectar packet spoofing, esta técnica consiste que ambas instancias del Neubot capturan los paquetes entrantes y salientes de la tarjeta de red entre ambos peers y son guardados en un archivo, luego las trazas son comparadas y se verifica que sean iguales, en el caso en el cual algún paquete no se encuentre puede ser debido a pérdida o a que el paquete fue cambiado en el alguna parte de la red antes de ser entregado al otro extremo. Una vez culminado el test peer-to-peer los datos son almacenados en la base de datos y los clientes vuelven a estar disponibles para realizar otra medida con otros peers.

Luego de los diferentes inconvenientes que han acaecido con diferentes proveedores de servicio sobre todo en Estados Unidos, la comunidad científica se ha interesado en desarrollar diferentes sistemas para tratar de medir la neutralidad de la red y recoger datos para evaluar los diferentes proveedores. Existen principalmente dos tipos de estrategias o enfoque, una basada en desarrollar soluciones para detectar problemas manifestados por la comunidad de Internet o de un proveedor en específico o desarrollar un sistema para obtener datos significativos que puedan evaluar el rendimiento o posibles acciones que los proveedores estén tomando para discriminar, bloquear o retrasar tráfico de sus clientes.

Contents

Chapter 1

Network Neutrality

1.1 Introduction

The Internet has expanded incredibly in the last years with users accessing new multimedia services as a well as common and established service. The Internet has been using a very simple and effective best effort traffic delivery, this decision was taken with the basis that usually data networks are used to deliver simple data services such as email, websites were the user does not need a quality of service neither timing restrictions in delivering the data. However, Internet is growing up and new services are constantly being added, consequently in the present we have access to Internet video, voice services, online gaming, were it is fundamental to have huge amount of data on continuous stream and in some cases, like voice and gaming, timing constraints. The demand for such services continues to expand and in consequence Internet Service Providers (ISP) are taking a new approach on managing network traffic from a "best effort" to prioritize network traffic to ensure a new quality of service.

The introduction of traffic prioritization may benefit consumers by delivering data faster and ensuring a quality of service. But in any case this change has led to a number of policy concerns and needs special attention because it implies to change the behavior of the Internet we are used to know.

The first concern adverts that the technologies needed by providers to prioritize traffic gives them too much power over the operation and access of the Internet. As a consequence we now have to think of a multi-tiered Internet, content providers (Google, Wikipedia, iTunes, among others) need to pay for different service levels [1], having the possibility to limit competition, specially for smaller content providers who cannot afford to pay for higher level access. On the other hand, ISPs can apply

control over the access, discriminating between different access levels.

A unified and clear concept or definition for Net Neutrality is very difficult to find, there is an open debate. But, a concise, maybe idealistic definition of Network Neutrality, can be the one given by Google [2], it states that: Net Neutrality is the principle that Internet users should be in control of what content they view and what applications they use on the Internet. The Internet has operated according to this neutrality principle since its earliest days. Indeed, it is this neutrality that has allowed many companies, including Google, to launch, grow, and innovate. Fundamentally, net neutrality is about equal access to the Internet. In our view, the broadband carriers should not be permitted to use their market power to discriminate against competing applications or content. Just as telephone companies are not permitted to tell consumers who they can call or what they can say, broadband carriers should not be allowed to use their market power to control activity online. Today, the neutrality of the Internet is at stake as the broadband carriers want US Congress's permission to determine what content gets to you first and fastest [21]. Put simply, this would fundamentally alter the openness of the Internet.

Besides the several ethical and legal implications of Network Neutrality, the main objective of this thesis is to analyze the Network Neutrality issue from a technical point of view, propose a design and implementation of a system able to measure relevant parameters to successfully obtain data to allow further work on studying the behavior of the broadband service providers and give the customer a simple evaluation of the performance of their connection.

The structure of the thesis is divided firstly, in a brief introduction on Internet and how is the Internet currently working. Next, an explanation on the main technology that is allowing providers to shape customers traffic, Deep Packet Inspection, a technology that has created with mainly security purposes to detect and avoid security threats, and is so potent that evolved to traffic management and traffic shapping. After explaining DPI, it's necessary to explain basic concepts on P2P to afterward introduce the developed system called Net Neutrality Bot (Neubot), beginning with the specifications and functionalities. Then the design and implementations parameters and technologies used for the deployment of the Neubot.

1.2 Internet: History and Operation

The Internet provides a Universal Service, just as the Public Switched Telephone System provides a Universal Service. However, Internet is offered as connectivity at

network layer using IP protocol, instead the PSTN also specifies all the characteristics at all levels like, delay bounds, minimum capacity, and availability. Generally Internet is designed to provide link speeds over-provisioned (exceeding the speed of current applications), but it is not part of the service or protocol specification, it is an implicit design agreement, nor is there a forum for agreeing what might be such a part of a service, since services and protocols are dealt with by different communities, unlike the combination offered by the ITU.

Recently, there have been some situations [12] where IP connectivity is available but higher level protocols and services have been affected by mechanisms that prevent access to those higher level services, generally filtering packets devices like firewalls, IDS and other security systems are used by providers to block malicious traffic, spam, viruses or DdoS among others.

The Internet is conformed by the interconnection of different networks spread worldwide [4], this networks are generally Service Providers that are interconnected. The presence or absence of routes in ISP routers determines whether a destination network is reachable. Each ISP maintains a set of customer routes for its directly attached customers. To reach networks (and hosts) outside its network, and thus provide value to its customers, the ISP must establish a business relationship with other ISPs to exchange routes and data traffic. Mainly there are two interconnection methods [3]:

- Peering Arrangement between ISPs: customers attached to the ISPs can reach both networks. A peering arrangement between ISPs has the following general characteristics:
  - Peering is performed at a public Internet Exchange Point IXP or via a direct connection.
  - Parity in size, geographic proximity (both ISPs are in the same region or country), and traffic symmetry (equal traffic volumes in both directions) is generally assumed between peering partners.
  - Costs are shared in that each ISP pays for its own router ports and the circuits necessary to establish the peering connection.
  - The arrangement is non-transitive, which means that ISPs will not exchange customer routes learned through peering with other ISPs.
  - Each ISP keeps their customer subscription fees to themselves. Assuming equal exchange of traffic; otherwise a significant imbalance in traffic might require a financial agreement.
  - Bilateral and multilateral (more than two ISPs involved) arrangements (MLPA) are supported.

- Transit Arrangement Model:

  Using peering between ISPs does not provide connection to the Internet, but that is where the transit model comes into play. In a transit arrangement, a larger ISP (typically a tier-1 provider) sells a smaller ISP access to the global Internet routing table that it maintains. The provider of the transit service to smaller ISPs is called a network service provider (NSP).

  An ISP pays for the cost of a circuit and a port into the NSP network; it can then advertise its customer routes to the NSP. In return, the ISP receives a default route pointing to the NSP. In effect, the ISP obtains complete access to the global Internet by becoming a customer of the NSP.

  A transit arrangement between ISPs has the following characteristics:

  - Customer ISPs incur the total cost of connecting to the NSP.
  - Transit service is prohibited over a public IXP.
  - Some NSPs offer local peering and transit services.
  - Smaller ISPs typically peer with other small ISPs at an IXP and establish a transit connection with an NSP to connect to the rest of the Internet.

1.2.1 IP Service

The Internet has been online for approximately thirty years, using IP protocols that interconnects nearly 1 billion worldwide devices today. It is conformed by a significant number of interconnected Service Providers, that allow the routing of packets using intra-domain and inter-domain routers to deliver packets from an end system to another end system.

The IP protocol design has a very important characteristic that gives an incredible simplicity, functionality, ease of implementation and the most important for the Internet is scalability, mainly due to the "stateless" characteristic of the protocol. But also the scale that the IP level currently has in the Internet is such important that it's become very ambitious and logistically challenging to introduce enhancements or modifications because it needs backward compatibility, to conserve connectivity, for a substantial period until migration is completely done. The most relevant example of evolution of IP protocol is the Ipv6 that is to add scalability to the addressing scheme used by the predecessor.

The core service model supports a very simple definition of performance, traffic has been delivered on a best effort basis, which is to say that there is none. Instead,

it is implicit in provisioning in each segment of the network (and varies with time, and with source/destination), what basic performance one might see. The evolution of the net has been focused on the core connectivity, and a low cost method to allow applications to coexist in a shared resource, based on congestion avoidance and control by end systems. The complexity of these overall developments generally shows up in two places:

- Below IP, mapping packets onto various link technologies.
- Above IP, in transport (TCP, RTP/UDP) and application (HTTP, P2P) protocols.

One of the key areas of evolution in terms of differences between ISPs has been that of Service Level Agreement SLAs. Many NSPs offer statistical guarantees of performance (above and beyond a simple bland statement of "Best Effort"). For example, zero packet loss is offered by some tier-1 ISPs, while 95th percentile delay guarantees are given by others [5]. Inevitably, there is a tendency to improve the service offers, as a consequence of tools for provisioning and traffic engineering become more widely available, and as capacity prices have continued to fall, making the feasibility of pure statistical multiplexing based guarantees easier to achieve (even for VoIP traffic).

Traditionally, the Internet has been treated as a fair network; but naturally some factors can demonstrate that the Internet Protocol Suite are not completely fair:

- End-to-end service: The majority of Internet applications use TCP as a transport layer. The TCP throughput is limited by some constraints. First, the bottleneck capacity, it can be your end or the far end's link speed or system I/O capacity. Second, the throughput is limited by any other user's TCP flows traversing a shared bottleneck. Third, the throughput is limited by the TCP implementation used as well as the TCP parameters, windows size, MSS and others. Finally, and most arbitrarily, the capacity is a function of the round trip time and packet loss probability on a link. Further you are from a sender, the less capacity you get than other connections.
- Inter-domain Routing: To reach a site on another service provider's network (the most common case), the traffic must traverse at least a border router. This introduces additional delay, but also, if the path traverses several ISPs, it maybe that the return path is not the same. This has a different effect on this traffic than others. This is a side effect of the business relationships between ISPs, they are using routing tables, they do not target the customer.

- Proxies Caches: they serve to distribute load, and improve users' experience in terms of delay (in fact, simply a precursor to p2p and torrent ideas). However, caches implement rules to control the performance seen by the overall set of users. Indeed, many popular news and software distribution websites now implement admission control algorithms to control the perceived performance. The net effect is that users during a request event, the customer can see messages communicating network congestion for instance.

1.2.2 IP Service: Evolution

The basic IP service has no real definition, simply best effort. However, many ISPs and some Internet Exchange Points define SLA. In circuit markets, you buy facilities to connect points with certain characteristics. The most common parameters that affect SLA are:

- Isolation: my traffic is not impacted at all by yours.
- Protection: my circuit is backed up to the nth degree by failover paths.
- Throughput: I get the capacity I pay for.
- Delay: whatever pattern of packet timings I send with is preserved at the far end, and I see non time-varying delay.

The generality of the Internet has led away from a purely TCP (and associated Best Effort tolerant) based applications. Now we have a very significant and growing number of users of network applications such as VoIP, IPTV, video-conferencing and networked games. Note that each of these applications has user expectations associated both with performance, and with being charged. We are not averse to paying for phone calls, for watching some TV programs, for being charged a lot for (legacy ISDN) video-conferencing, or for paying to be in a game (or even for objects in the game). Internet users now expect to see some of the properties of circuit switched networks.

A number of technologies have emerged to support services that look a bit like circuits in the Internet, although most are only deployed within a single ISP, and often, mainly for corporate customers so far.

- Differentiation: The IETF community has been struggling with a variety of concepts for introducing Quality of Service mechanisms to the Internet for long time. Finally, we have a simple, but effective technique, which some ISPs have deployed, principally to support the legacy services on IP such as VPNs and VoIP backbones for a national telephony service. However, these are good

proofs of concept and there are plenty of customers for a more dynamic service enhancement.

- Provisioning: Any technology for QoS assurance of any kind is deployed coupled with a detailed knowledge of the topology of the network, the workload and traffic matrix, and its variation over the day, and a detailed model of all source behaviors. These are then fed into some provisioning model which also contains the traffic engineering mechanisms that the ISP is deploying. This could be based on a tool such as network algebra, or an emulation or simulation that is used to compute whether a new user or service can be admitted. The timescales of this are rather different than what was used in traditional admission control for telephone calls, but that is because we have more headroom in today's networks, and we have better tools to comprehend aggregate behaviors in the core.

Piecewise deployments to offer new services can be seen as potentially applicable to other changes to the core IP service model, such as:

- Security: some ISPs provide firewall services in addition to NATs to protect users from unwanted access. In some cases this may go further and include black-holing of sources of SPAM, and of DDoS attacks (sometimes only on request). The idea of providing more sophisticated security services (e.g. signed/authenticated and approved system distribution for sites) is already common place in private networks, and one can imagine ISPs requiring (and providing) approved systems and system patches to remove vulnerabilities.

- Mobility: The last few years has seen the emergence of Wireless ISPs (WISPs), offering pay-per-use wireless hotspots. Quite a few of these provide roaming arrangements, whereby credit on one service can be used on another.

- Multicast IPTV: is starting to take off with content problems being resolved, and net performance finally exceeding the threshold necessary to offer reasonable quality realtime TV. However, some live events may be of primary interest to large groups, and we may see pay-per-use IP multicast finally take off. On the other hand, P2P TV is also emerging as a model which doesnÕt stretch the ISP at all, but meets the requirements provided enough up-link capacity is available from participating customers. The ISP might in either case, broker the content and rights.

1.2.3 Access Networks

An entirely different version of the net neutrality debate concerns the access network. This sort of debate can also be held concerning wide area wireless (cellular)

access, and has been noted in the previous section, it could also apply to WiFi pay-per-use hotspots [5].

Legacy services with vertical bundles (PSTN, with phone line which happens also to be the last mile access for IP, same for cable TV) are crucial to many users of the Internet. The operators who own these local loops are quite heavily regulated in many parts of the world, in terms of telephony, and in terms of allowing competition access to the exchange (or head end in the cable case) end of the lines. Whether the line/access are bundled or unbundled is crucial.

The costs associated with maintaining 100s of millions of phone lines are quite high. The cost of deploying ever increasing speed DSL kit at the exchange ends is also high, and many incumbents would like to offset this by increasing charges. The cost of providing an alternative is also high, although fixed wireless broadband is a possibility for near future, as is the replacement of the entire access net with fiber in highly developed parts of the world.

However, if the operator that owns the last mile also still owns significant long haul networks, and wishes to capitalize on both, there is a strong incentive to provide some modest level of tiered Internet, by offering improved access link speed, provided some bundle of higher levels is subscribed to. This is a similar approach in comparison with telephone, digital TV and cellular telephone services.

Chapter 2

Deep Packet Inspection DPI

Deep Packet Inspection represents until now the top of the chain in packet filtering techniques. Beginning with simple packet inspection, used to identify packets in a firewall and decide wether to forward it or not at IP level. Based on specific rules that packets must match and then an action is taken (drop or forward it). The rules only verify that some field within the header has an specific value. For instance a firewall can examine for packets incoming from a specific IP address and decide not to forward them.

Further evolution in processing capability brings on stateful packet inspection. A firewall architecture with the main scope of tracking TCP and UDP connection keeping a state table for each connection; controlling that each packet must be part of an already established connection. Otherwise it should be an establishing connection packet like SYN for the TCP case, in that case it has to build a new entry in the state table. If a packet does not match with an entry in the state table it is discarded.

Introduced as a technique to monitor and shape traffic, Deep Packet Inspection extends the behavior of the predecessors adding the functionalities of other security techniques like intrusion detection system and intrusion prevention system. DPI analyze the IP packet, and decide to forward it or not based on static rules matching. They compare the data inside the packet payload to a database of predefined set of signatures, using standard application protocols signatures to identify and track them and predefined attack signatures (a string of bytes).

The comparison of the payload with the predefined rules is a very expensive task in terms of CPU usage because its necessary to look through multiple byte patterns that must be verified. Also a requirement that increases the complexity in design and implementation is that searches have to be done at wire-speed to avoid slowing the network.

To give an overview of the utilization and capability that DPI brings to the network, imagine a sort of firewall that is able to detect and decide to forward or not P2P traffic, e-mail, Web, Web applications, XML based protocols, VoIP, potential viruses and others. The decision is taken in real time, or at least as fast as to avoid slowing significantly the network performance. The actual capacity devices that are in the market can manage to inspect a million simultaneous connection at 10 Gbit Ethernet speeds.

2.1 Network Neutrality and DPI

Now having a slight idea of the power of DPI, ISPs are able to gather big and significant quantities of information about the customer traffic using this technology in their networks. Bringing the possibility to apply new traffic management techniques capable of blocking, throttling, and prioritizing traffic with more detail parameters, i.e. protocol, and application.

A feasible application can be illustrated by an ISP that agrees with a website to give higher priority to the traffic incoming to their customer from that website, this means that customers using that website are going to have better speeds than using others, leaving others websites with the best effort Internet basis but knowing that there is a higher tier that goes faster.

Another weak point that necessarily has to be mentioned, is that identifying the different services that their customers are using, ISPs can block or degrade services that they disapprove because of traffic congestion or any parameter established for "traffic management" policies by the ISP.

2.1.1 Inspecting Packets at different levels

Before DPI it was not possible to identify modern Internet applications that are designed to pass through home firewalls and NATs. These applications use different methods like, not using constant ports, or encapsulate the traffic in other well-known protocols like http. Then, identifying the ports is not enough, nowadays we have almost all kind of applications on a Web browser, the users are able to have audio, video, data. And all that content is requested to the same port.

The information an ISP can obtain using DPI is invaluable, in terms of the OSI layer model they have information from layer 3 up to layer 7. Being the layer 7

the most important obtained data, with the expectation of obtaining how the applications are negotiating the connection in terms of ports, how they traverse the firewall by connecting to an intermediate server continuously keeping track of the client. Or even deciding wether to slow down the request of a news video embedded in a website because the network is congested and it is necessary to prioritize the e-mail.

In the application layer we have the actual data that applications sent to the lower layers to send across Internet. Some common applications are Firefox (web browser) BitTorrent clients, like Azureus, or VoIP clients, like Skype. DPIs devices obtain the application data by removing the headers, then the payload obtained is used to determine the application used. There are primarily two approaches to identify the application, seeking in the user data or trying to match some string or particular known string of bytes.

Procera, a DPI fabricant affirm that they are able to detect more than three hundred application protocol signatures including the most popular like BitTorrent, HTTP, SMTP, SSH among others [8]. Another vendor claims that they can identify more than the protocol, specifically detecting particular multimedia content encapsulated or carried by HTTP like YouTube and Flickr.

To analyze the signature of the protocol application data several techniques can be used. Discarding the port analysis because applications are constantly changing ports and in consequence is significantly inaccurate. The first approach is string matching but not all applications use a string to identify the protocol. Also, using patterns with numerical properties represent a good option by determining the payload length or specific response sequences. The next approach can be considered very dangerous because it brings out a more deep privacy concerns, the DPI assemble all the packets and returns legible evidence of e-mails, websites. Being the most intrusive approach because they are obtaining content and information from the customer data.

Some of these things can be done by looking at a single packet, but many cannot. DPI gear can generally extract information from traffic that varies by application type: IP addresses and URLs from HTTP traffic, SIP numbers from VoIP calls, filenames of P2P files, and chat channels for instant messages. Grabbing this information requires a look at a whole set of initial packets until the necessary information is gained, referred to as examining the "low".

2.1.2 Internet Service Providers and DPI

With the overview of the potentials that DPI has this technologies can be applied in a highly granular fashion. The DPI rules can be created to treat each customer data in a different way, and depending on the rules that application, service or protocol can be shaped, blocked or give a better priority. Until now the Internet has been based on flat Internet access, without data distinction and based on bandwidth or a combination of bandwidth and quantity of data.

With the introduction of DPI introduces newer service tiers to access the Internet being infinite combinations among applications. For instance a customer can purchase a service to access web (HTTP) but not P2P applications, if the user tries to use a BitTorrent client it is not going to work as the customer requested [12]. As this case we can think of much more combinations including gaming, VoIP, video conferencing.

2.1.3 Internet "Overloading"

The idea here, from the perspective of the DPI vendors, is that the Internet now generates and streams more data than the current transmission network can handle without shaping or throttling [7]. Given the situation with P2P users and streaming video watchers, this sort of content alone could cause delays for content that is arguably more critical and time-sensitive for an ISP's customers than an illicit Hollywood release: e-mail, instant messages, traditional web browsing.

Seen in these terms, the DPI vendors argue that ISPs which "do nothing" to shape traffic on their networks have actually made a choice. In this case, the choice is in favor of chaos and bottlenecks at peak periods. No matter how much bandwidth is currently thrown at the problem, P2P, Usenet, FTP, and streaming video will fill it [7]. Handling this overflow of data, surge responsibly means using traffic shaping, at least during the periods of highest use.

2.1.4 Network Neutrality Concerns and Related Points of View

Now, this entire approach to managing traffic doesn't sit well with some associations who call for neutrality on their networks. Recent research has shown that a nondiscriminatory network will in fact require up to twice the peak bandwidth of a tiered and shaped network, but this doesn't necessarily mean that this is the more

expensive approach. Because simple over-provisioning can be cheaper in the long term instead of investing in implementing DPI with all the resources to maintain and monitor it.

The debate is made complicated by the fact that "network neutrality" has a hundred differing definitions.

For a thoughtful definition, consider the one given by Daniel Weitzner [11]. He states four points that neutral networks should adhere to:

- Non-discriminatory routing of packets.
- User control and choice over service levels.
- Ability to create and use new services and protocols without prior approval of network operators.
- Nondiscriminatory peering of backbone networks.

Savetheinternet.com [10] has spearheaded the network neutrality drive in Congress, and it has a shorter definition available: "Put simply, Net Neutrality means no discrimination. Net Neutrality prevents Internet providers from speeding up or slowing down Web content based on its source, ownership, or destination".

If that's not clear enough, they provide an example. "When we log onto the Internet, we take a lot for granted. We assume we'll be able to access any Web site we want, whenever we want, at the fastest speed, whether it's a corporate or mom-and-pop site. We assume that we can use any service we like: watching online video, listening to podcasts, sending instant messages; anytime we choose".

This kind of definition as well as the Google one mentioned in the Introduction are very generic and ISPs and DPI vendors argue that this sort of "never discrimate" policy isn't much more than unworkable idealism. Such a network will in fact fill up with data; companies that don't apply any filtering or shaping packet flows strategies are accepting that "circuit oriented" services like VoIP, videoconferencing, and online gaming to get slow and suffer interruptions, to get delayed as BitTorrent and YouTube packets overload the network. Downloading a large video file, is hardly the sort of application that is mission critical, and few customers are going to abandon ship because their YouTube videos take an extra two seconds to buffer. But customers do care if their VoIP service consistently goes glitchy or has tremendous lag, or critical e-mails and IMs are delayed in transit.

Where you come down on these questions may vary depending on where DPI gear is deployed; many people have less problems with its use by last-mile ISPs who interact directly with consumers. Throttling P2P traffic to keep the network open for other uses might be fine, but the concern is magnified when such gear is rolled out by the backbone operators. With access network ISPs, at least customers have some options for switching if they don't like the terms.

But there are so few backbone operators (NSP), and they wield so much power, that a deep concern on a net neutrality perspective is if backbone providers start looking at Google and propose, "If you want decent transport over my pipes, then you have to pay my toll". That's because there's no way for the end users to select which NSP; all of the users of the broadband ISPs who are downstream from that backbone will see their access to Google start to suck, but there's not much they can do about it because it's not really their ISP's fault. In other words, the backbone providers have a more insular, more monopolistic, non-consumer-facing position in the Internet hierarchy, so if they decide to ditch neutrality and start squeezing websites and online service providers, then there's not much that can be done.

These are deep waters, and there are complex arguments to be made here (for a detailed engineering discussion of the issues facing "best effort" routing on a congested network, take a look at this IETF Internet-Draft by Sally Floyd and Mark Allman). DPI gear makes plenty of objectionable behaviors possible, but it also opens the door to network virus scans and DDoS defense mechanisms that could do real good. By making it possible to purchase access only to the specific services or protocols that one needs, DPI could also make the Internet cheaper for casual web and e-mail users. Like most technologies, the gear itself enables a great range of uses, and it's up to the operator to be responsible.

One of the grand challenges to net neutrality was the subject of many of the companies representations to governments, and that was the threatened actions by some ISPs to block or lower performance to certain applications en masse. The statements made by some ISPs implied that overlay services that are crucial to many users such as VoIP and Web Search engines (specific examples of course being Skype and Google) were generating big amounts of traffic in their networks for free.

This emotive term was used almost certainly by marketing people, since it has connotations of illegal file sharing and piracy. However, most large scale overlay systems buy significant quantities of Internet access at very high speed, and (more importantly) buy it from many ISPs in data centers in POPs (as discussed in the previous section) so that they can offer a global application service. Let us think about that for a bit because it is really quite amusing. An ISP is not forbidden from

also being a content service provider, they provide sites with different channels and communities. An ISP that has data centers could build its own VoIP call-out service, and its own search engines. Indeed, it might be able to pinpoint "click-through" far more accurately than a search/lookup service at lower cost simply by monitoring network access patterns.

Chapter 3

Network Neutrality Bot: NEUBOT

3.1 Introduction and Basic Concepts

NEUBOT, conceived by the NEXA Research Center in May 2007, is a distributed system; it is aimed to measure quantitatively characteristics like latency, bandwidth, jitter, packet loss rate and filtering of specific IPs port/applications, web pages or web services, on specific network segments, to determine if there is any degradation of traffic quality of a specific ISP.

The basic architecture of NEUBOT consists of a client application that end-users, who voluntarily participate in the project, can install on their computer. The application is designed to perform measures with two different approaches: the traditional client/server approach mainly is in in charge of characterising some basic end-to-end parameters of the user connection. On the other hand, Neubot is capable of using the advantages of a peer-to-peer topology to measure some protocol specific performance.

Considering the uncontrolled nature of the environment, some precautions must be taken. The client application, for instance, must be able to detect the network load generated by the client, particularly in the case of applications that consume significant bandwidth, such as peer-to-peer ones. In addition, the client application has to be able to ensure basic security mechanisms to provide a satisfying degree of anonymity, including by protecting the IP address of the client.

Another important security point is authentication; both parts (client and server) must be authenticated to avoid server masquerading, implying that the client would

send information to a fake server. Authentication of the client is also necessary to prevent the server from receiving false measurements from distrustful sources. To implement the two-way authentication we can establish a Secure Socket Layer, using Digital Certificates for clients and server in order to authenticate both sides and establishing a session key to encrypt the data passing through this state-full connection. Establishing the SSL connection Data Confidentiality is guaranteed because all the data is encrypted with standard algorithms like AES, DES or 3DES. With respect to data integrity, it is reasonable to use one-way hash functions over the transmitted files between client and server, making it possible to verify that the data has not been modified during the transit over the network.

The server is separated into different logical instances, the measure server is in charge to implement the techniques to measure the user connection characteristics. Also it can implement a peer to perform the protocol specific test. Finally the server includes a database to store all the gathered data. The server must verify the identity and the integrity of the measurement files, classifying the different data according the ISP of the client and establish if the values expressed are valid and have not been intentionally affected by some factors intentionally (we need a criteria to discard accurately the affected values). Among these criteria, the client program can measure bandwidth usage before starting to take the measures - on the basis of used bandwidth and the bandwidth measured for the NEUBOT client it is possible to determine approximately the degradation of the service due to the user traffic. Another approach can be suggesting the user to close all the applications that are using a significant amount of bandwidth and the measures should not start until the user closes the application. Precise identification of these criteria is currently under investigation.

3.1.1 Functional Specifications

The functional specification is described in the following table where it's described main elements are shown, including a brief description of the element and the principal function. The idea is to state the main objectives and functions related to an overview of the system, stating the main system blocks necessary to build a distributed system able to characterize broadband connections and protocol specific performance:

Element	*Description*	*Related elements*
Network Measures (Client)	This element represents the core of the system because it is in charge of taking the necessary meassurments with the different sites and different parameters. To begin, these are the parameters to measure: • Reachability: using ICMP protocol. Echo request messages. • Latency & Jitter: Establishing a RTP flow and looking into the time stamps in the sender and receiver, calculating an average. • Bandwidth (Throughput)	Update
Network Load	This element has to be able to determine the bandwidth used by the client in order to determine if the measures can be done. Also if it is possible to know the applications running in the client it can suggest to the user to close those that can be bandwidth consuming like p2p applications.	Network Measures
Connection (Client/Server)	Using TSL/SSL protocol to authenticate and establish a secure connection between client and server. This element also needs to handle the necessary operations involved in the connection between client and server like send data and receive data.	Network Measures
Update (Client)	The update element consists in having a flexible design to pass the parameters to the client, the parameters are the sites, IP/ports, URLs to measure. This can be done by having a file on the server and checking the version that has the client, if it is necessary the server sends the file, so the client has updated the parameters.	Network Measures
Creation of the Measure File (Client)	After having all the measures from the Network Measure module it is necessary to collect all the information and build a file to store in the server	Connection (Client)
Receive Measures (Server)	Parse the data and store it.	(Server)

Table 3.1. Functional Specifications

3.2 UML Diagrams

3.2.1 Use Case

The following diagram represents the typical Use Cases of the Neubot System. There are two main actors: the External Sites, that are different websites and servers to perform the measures and with the Database that constitutes the measures data repository. Basically, considering the main requirement of obtaining some network parameters and measures, the System has been thought to perform different operations from the design point of view, in order to also cover the characteristics of a

distributed system.

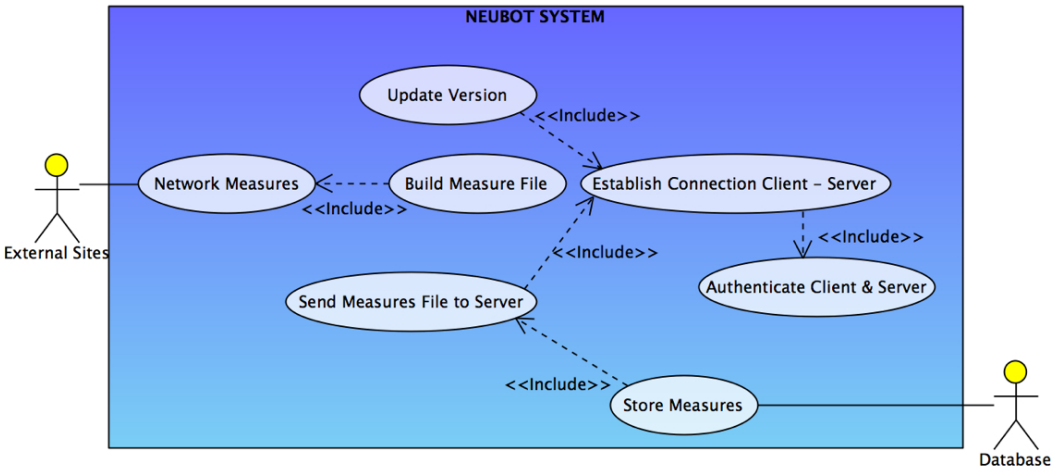

Figure 3.1. Use Case

3.2.2 Activity Diagram

The activity diagram shows the high-level actions the system should do. A process diagram flow oriented allows to design a very linear and systematic path, because there is no such interaction with the user (in the client side) that implies asymmetric behavior, and the actions are followed strictly in sequence, in order to have a lightweight application that does not consume much resources to avoid disturbing the user when the Neubot is running.

3.2.3 Class Diagram

The class diagram design is based in modularity and scalability to allow the System to grow easily by adding new features and modules.

3.3 Security and Authentication

As mentioned before, security and consequently authentication have a key role in the specifications of the system, because we have to be sure that the gathered data and measures are reliable and real. In a future stage of the project when some data is analyzed to verify a performance problem with an specific client or ISP the data must be trustworthy.

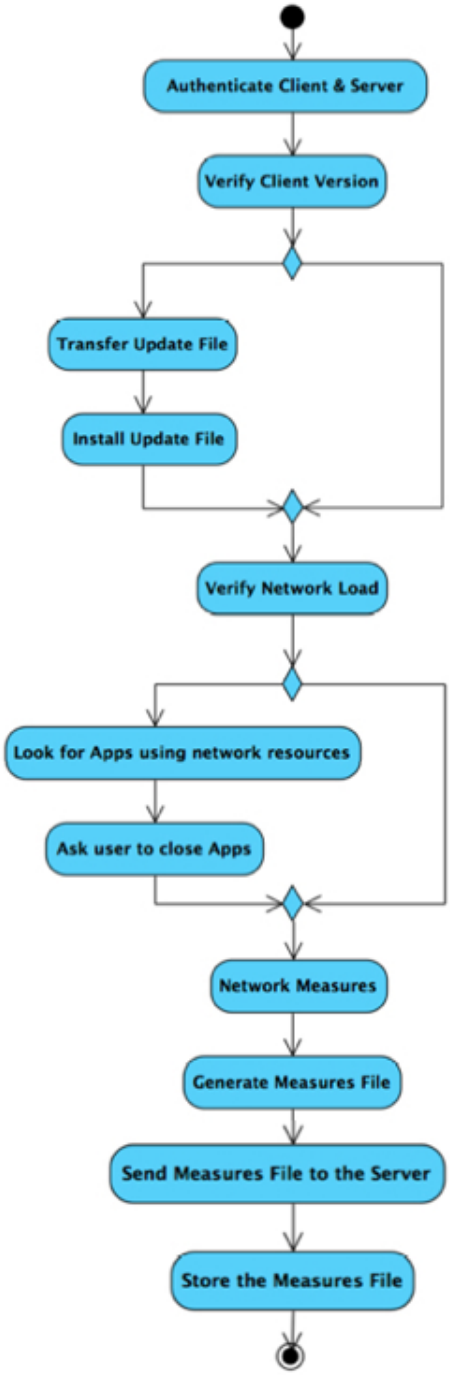

Figure 3.2. Activity Diagram

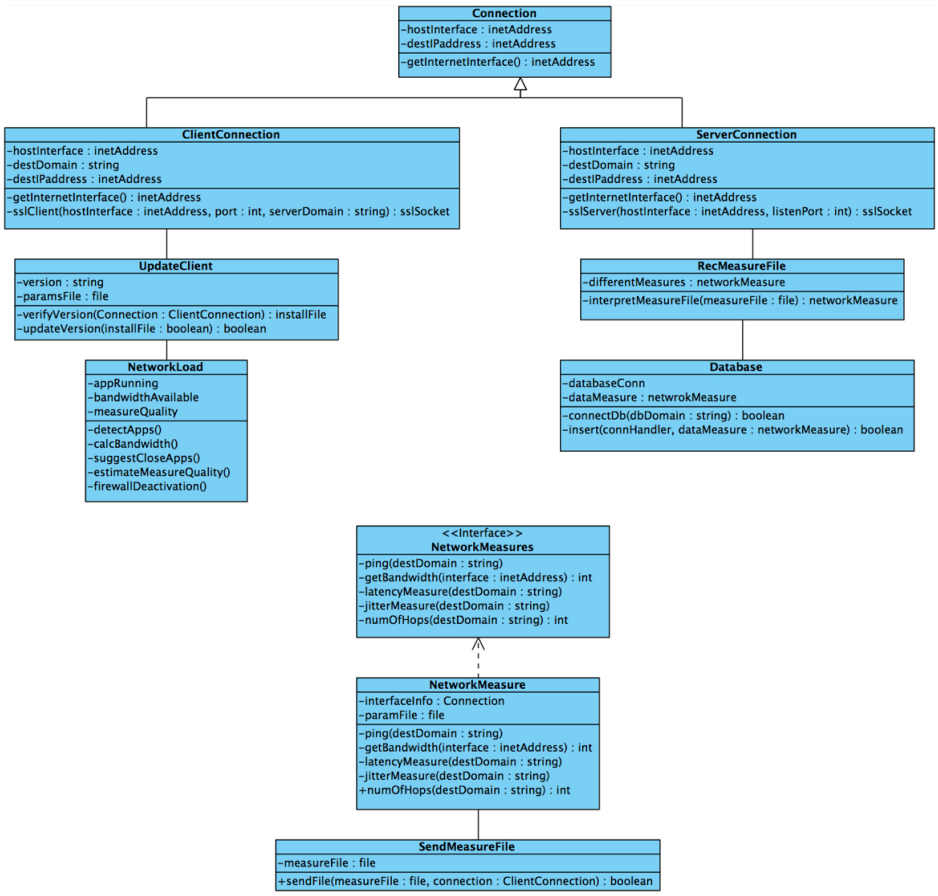

Figure 3.3. Class Diagram

3.3.1 Database Server Side

The Database must provide information security to avoid undesired activities over data. At application level, authentication and authorization services must be implemented to control the use of a single sign-on service, that in combination with a log service in charge of recording users activity can trace irregularities. At network level the Database can be located behind a firewall to avoid attacks from outside the internal network.

3.3.2 Client Side

Security in the client side is limited to the communication between client and server or other peers in the Internet. To avoid masquerading attacks from both sides (client

or server) we must provide two-way authentication through using Secure Socket Layer (SSL), using Digital Certificates for clients and server establishing a session key to encrypt the data passing through this statefull connection. Establishing a SSL connection Data Confidentiality is guaranteed encrypting the data with standard algorithms like AES, DES or 3DES. With respect to data integrity, it is reasonable to use one-way hash functions over the files transmitted between client and server, making possible to verify that data has not been modified during the transit over the network.

3.4 Database Creation and Management

In this section some of the minimum needs of a database system are stated in order to gather the measurement data accomplishing the most important characteristics of a distributed system.

3.4.1 Upload of measurement data onto database server

In order to connect to the database with the Neubot client, the proposed method is to use a database driver (usually we have a database driver for most common database platforms). Using an Application Programming Interface the Neubot client can connect directly with the database to insert data corresponding to measures. A database driver simplifies the development of another server to collect data from the client to insert it in the database.

3.4.2 Client Anonymization

To anonymize the addressing information of the customer used on the measurement data, the IP address of the client should be changed in a way that we can maintain the necessary prefix to obtain the Service Provider but give no information about the specific address used on the measurement. The proposed method consists in conserving the network address of the client and simple live in blank the client part. Another option is to apply an encryption algorithm to change the IP address but be able to resolve the real address in case it is needed in the future, to do this the encryption function needs to be reversible.

3.4.3 Database Integrity Check

The database integrity check is going to be in charge of the platform decided to implement the database with.

3.4.4 Database Redundancy

Database redundancy gives the system more reliability, more tolerance and better performance from the client-side point of view. However, it is difficult to implement due to logistics. This necessity depends on how the system grows and the quantity of users served. Two types of redundancy are possible:

- Local Redundancy
- Geographical Redundancy

3.5 Measurements: algorithms and techniques to compute/measure specific performance indicators

In data networks, bandwidth represents the data rate, throughput, at which a network link or a network path can transfer. The concept of bandwidth is fundamental to packet networks, is related with the amount of data a link can deliver per unit of time. For many data-intensive applications such as file transfers or multimedia streaming, the bandwidth available for an application directly affects its performance.

Bandwidth is also an important factor in several network technologies. Several applications can benefit from knowing the bandwidth characteristics of their network paths. For example, peer-to-peer applications form their dynamic user-level networks based on available bandwidth between peers. Overlay networks can configure their routing tables based on the bandwidth of overlay links. Network providers lease links to customers and usually apply charges based on bandwidth purchased.

There are several considerations related to the term bandwidth that need to be clarified to precise the different throughput metrics. The most representative bandwidth metrics are: capacity, available bandwidth and bulk transfer capacity.

3.5.1 Capacity

Capacity at layer 2 generally is related to the possibility to transmit at a constant data rate that is limited by the physical layer, including the limitations of transmitter and receiver hardware and the bandwidth of the physical medium. Concerning the IP layer the capacity is reduced due to the protocol overhead of layer 2 framing.

3.5.2 TCP Throughput and Bulk Transfer Capacity

Another important bandwidth metric is data rate or throughput of a TCP connection. TCP is the most used protocol in the Internet, carrying almost 90 percent of the traffic. TCP throughput metrics are more interesting for end users, because it measures the actual data rate that they can achieve [13].

Unfortunately, it is not easy to estimate the throughput of a TCP connection. Various factors may influence TCP throughput, including transfer size, type of cross traffic, number of competing TCP connections, TCP socket buffer sizes at both sender and receiver sides, congestion along the reverse path, as well as the size of router buffers and capacity, and current load of each link in the network path. Variations in TCP specification and implementation, such as NewReno, Reno, or Tahoe, use of selective ACKs (SACKs) vs. cumulative ACKs, selection of the initial window size, and several other parameters also affect TCP throughput.

The BTC (Bulk Transfer Capacity) defines a metric that represents the achievable throughput by a TCP connection. BTC is the maximum throughput obtainable by a single TCP connection. BTC depends on how TCP shares bandwidth with other TCP flows, while the available bandwidth metric assumes that the average traffic load remains constant and estimates the additional bandwidth a path can offer before its tight link is saturated. To illustrate this point, suppose a single-link path with capacity saturated by a single TCP connection. The available bandwidth in this path would be zero due to path saturation, but the BTC would be about C/2 if the BTC connection has the same RTT as the competing TCP connection.

All the measures mentioned previously are important because different aspects of bandwidth are relevant for different purposes. An important issue is how to measure these bandwidth related metrics on a network link or an end-to-end path. Even so, the users are mainly interested in estimating the bandwidth of end-to-end paths to determine the actual service that they are receiving from the provider.

Neubot is designed to measure BTC of an Internet connection. Generally, the Internet access connection are dominated by two technologies: xDSL and Cable, representing the bottleneck of the Internet access. Measuring BTC we offer the possibility to gather the actual service obtained by the user, specifically related to the server they are connecting to, in order to offer a clear view of the Internet access speeds with different providers.

3.5.3 Techniques to measure characteristics of the broadband connection

Based on the previous analysis of bandwidth metrics we can state three basic parameters to measure in a broadband connection. Primarily the throughput to estimate the real data rate the user is obtaining with TCP and UDP. Then, delay related parameters to estimate the performance of real time applications, where its important bandwidth and low end-to-end delay. The main parameters that Neubot is planned to measure are:

- Throughput: with TCP and UDP transport protocols
- RTT
- Latency and jitter

3.5.4 Techniques to measure the performance of specific protocols

Measuring broadband connection parameters only give us information about the low level performance to compare it with the offered service. With the entry of DPI we need to obtain information of different services and protocols in top of IP layer in order to obtain a broad perspective of the performance of the ISP. To obtain information related to specific protocols and/or services it is necessary to implement different protocols and measure protocol-specific indicators. The first and most important indicator is ability to start/complete transactions (i.e. in a VoIP protocol it is mandatory to measure the correct establishment of a call using a signaling protocol and the correct exchange of the media using a transport protocol). Using the start/complete criteria only gives us the certainty that the protocol is being blocked by the ISP. In addition other mechanisms based on timing should be implemented to determine if there is a service degradation. The main protocols and services that should be measured are:

- VoIP
- BitTorrent
- Skype

3.6 Protocol specific measures via client/server and/or peer-to-peer architecture

For the performance specific protocols section it is necessary to develop an application protocol to synchronize and establish the communication between both ends of the system, who have to execute the measurements, in order to communicate which measure is going to execute, the status and the successful end, then negotiate the next measure to be made. A good alternative could be using a well known protocol already implemented, facilitating the deployment of the project, but generally this protocols are application oriented, as the popular file sharing protocols. As a result we decided to use a framework based on XML messages to implement the necessary protocols, providing flexibility and making services fit our purposes in a much simpler way than a file sharing application.

Clients-POLITO Server

This situation represents the most simple and immediate Neubot implementation.

Clients - a number of geographically distributed servers

Having the possibility to include several servers distributed geographically, it offers a better distributed architecture, replicating the services, and having the ability to obtain different measures at different distances from the same client, evaluating the performance from different "points of view". This solution requires the collaboration of institutions or individuals to have available servers.

3.6.1 Peer-to-Peer measures

The measurements are performed between clients, connected by a lightweight p2p protocol.

Clients-clients connections

For this architecture it is necessary to develop a P2P network satisfying the basic properties needed to making possible a communication between two peers in the Internet. The basic function is that peers should be capable to locate other peers through a presence service, then determine if the user is either ready to communicate and attend the requests. After peers localization some routing mechanisms are necessary to reach peers, as well as a protocol that after established a connection between two peers allows the negotiation for the measurements. This protocol can be similar to the one deployed for the Point-to-Point architecture. It is important to

consider that in this case the configuration of firewalls and NATs must be handled specially for some protocols like SIP that cannot handle this situation. Another important factor is the port scanning module, because with a physical/software firewall it could not be possible to achieve.

Hybrid P2P

After describing the client server model and the P2P model it is relevant to take into account a Hybrid architecture where the system can have a special peer (it can be POLITO) to discover peers and offer a mechanism to handle NATs and firewalls which are very common devices nowadays in xDSL clients using a Wireless Access Point to access Internet. Also this special peer can implement some special measurement modules like scanning port where OS or firewalls make it difficult to manage with other peers within the Internet.

Implementing the basic and necessary services of the P2P network as discovering peers and sending and receiving messages, the services for synchronising and negotiating the measurements between the peers can be built upon the P2P network. JXTA is a scalable framework for developing P2P networks it offers the core services and provides the interfaces to build services like file transfer. The idea to use JXTA is to support peers intentioned to contribute the peers to find them in the Internet and establishing the roles and modules to measure.

3.7 Techniques to minimise impact of user activities on measurements

Depending on the architecture we need a technique to minimise impact of user activities on measurements. For the case of the client server model, having the measure a limited time of execution where the user can be informed about the progress of measurements and the remaining time. The idea is to detect the network load and the applications using the network interface. If the load is considered as relevant, the application must ask the user to close P2P applications or file transfers until the measurements are done, otherwise the user must be encouraged to execute the application when the network load is low.

Alternatively, for P2P or hybrid architecture, the client application must run while the user is connected to Internet. When the application finds a peer able to do the measurement, the network load is measured and depending on the result the user is asked to close the applications consuming throughput. If the user does not want to close the application the measurements can be done by adding the conditions

that affected the measurements. Also the user can be prompted for the time where the measurement can be done and inform the other peer to see if it is going to be available. This can be done because the measurements can be repeated while there are peers on line. With this behaviour the users are encouraged to continue running the application.

3.8 Techniques to minimise impact of measurements on user activities

As well as minimizing the impact of user activities on measurements, minimizing the impact of measurements on user activities with the client server architecture, the user must be advised that he should reduce or contain the network traffic while the measurements are done. For P2P or hybrid networks we have more flexibility giving more options like asking an amount of time to retry. The other peer can be informed about this time and reserve it when available.

Chapter 4

NEUBOT implementation

To implement Neubot, we decided to use the best of the two most used network approaches, client/server and peer-to-peer. The client/server part is dedicated to characterize the broadband connection of the client and the P2P part has the main task of measuring protocol specific parameters. Both parts are wrapped in the same application. In this section I'm going to describe the implementation and the functionalities.

4.1 Client/Server Architecture

The client/server part is in charge of measuring basic parameters of the end-to-end connection between the client and the server. We decided to implement the client/server approach, in contrast to peer-to-peer, to develop this measures, because it is necessary to have a server with some basic conditions to obtain valid measures. Consequently, the tests are done with a server in a fixed location with some standard characteristics.

The most important characteristic is a connection throughput that can be at least equal or higher from the measured connections, generally home broadband connections speed range from 1Mbps to 20Mbps. Also it is necessary to have an always available physical server. The server is developed as a multithreading server, creating a thread for each received connection, in order to be able to attend several clients at the same time. The server also has the possibility to manage the different requests that are sent from the client called, transactions. The transactions managed by the server and their functionality are described in the following.

4.1.1 Transactions

The server handles different transactions, each transaction represents a different measure. Before the client/server reaches to the transaction phase, the client connects to the server establishing a TCP connection. In this TCP connection the server opens another random port and notifies the port to the client in order to connect to it. This way the already established connection is used as a control connection to send and receive the transaction related codes and instructions and in the new connection the actual traffic related to measures is transmitted. The traffic and resources consumed by the TCP control connection can be ignored, because the exchanged data is extremely low and generated when Neubot is not acually measuring.

RTT

The RTT is measured at TCP level. To implement this measure it is necessary to exchange messages between the client and the server as quickly as possible, therefore a simple small message is exchanged. When the server receives the request it takes a time-stamp. Another time-stamp is taken when the server receives the reply message from the client. In this way we obtain the round trip time of a message in an established TCP connection.

TCP BTC Download

As explained before, BTC represents a good measure to obtain the real data rate perceived by the user, affected by several constraints and factors of BTC. To implement this measure the server sends a fixed data quantity to the client. The client takes a time-stamp the moment before beginning to receive the data and takes another time-stamp after finishing the reception. Thus, having the received amount of data and the necessary time we can obtain the throughput of the connection.

TCP BTC Upload

The same methodology as explained before is implemented to obtain the BTC. The only difference in this case is that the data is sent from the client to the server, the time is measured in the server and then sent to the client to inform the upload BTC result.

UDP Download

This measure is called UDP and the goal of the UDP measure is to obtain the UPD BTC without the implementation of any control mechanism. The data is sent in

datagrams without any confirmation and the client needs to measure the time and count the quantity of received bytes. In this case we could have NATed and/or firewalled connections. The measure was tested with different kinds of devices and the result was not consistent, normally it could be able to traverse NAT and firewall not blocking UDP connections because first we measure the Upload BTC to open the port and then the Download. It probably depends on the vendor implementation of the NAT and/or firewall.

After finishing the tests the client establish a connection with a Database Server (in this case a MySQL server) to store the data obtained from the client/server test. The data is inserted as a new entry to the table and the connection is closed.

UDP Upload

The UDP upload is measured before the Download in order to avoid possible NAT/-firewall in the client side connection, in this way it is possible to open a port, but this is not a definitive solution to avoid NATed/firewalled connections, it depends on the vendor implementation and configuration of the device.

4.2 Peer-to-Peer Architecture

In this section the idea is to introduce how the protocol oriented measures are implemented, but before it is necessary to explain some peer-to-peer basics to be able to deploy a P2P application in order to introduce the implementation the Neubot P2P architecture.

In the HTTP/Web paradigm connections are established with one location that aggregates information from other sources. Then, the transaction is performed with the website, which coordinates the transactions among multiple clients.

In contrast, P2P implies direct exchanges: "If you have something I want, I go directly to you and obtain it" [15]. Likewise, I may have something you want, in which case the process reverses. In many cases, the exchange is mutual. The "something" could be a file that contains a picture or a sound or real-time information. It could be a service such as a storage capability or file conversion [18]. The beauty of conducting exchanges amongst peers is that the participants view each other as equals, and roles can shift as the need requires. There is no permanent dividing line between client and server. Unlike using the Web, P2P makes it easy to publish as to consume, to create as well as enjoy.

4.2.1 P2P and the Internet

The Internet is the ultimate peer-to-peer mechanism. The user can easily reach and interact with virtually any computer. Actually, the Internet started as P2P, a small group of researches wanting to share information agreed on protocol standards, and linked via a network [15]. Within today's definition, this network was almost pure P2P. The ultimate potential of the early Internet when unheralded due to the small number of participants as well as the technical jargon that obfuscated the network's communication value.

Most Internet services are distributed using the traditional client/server architecture. In this architecture, clients connect to a server using a specific communications protocol, like the File Transfer Protocol (FTP), to obtain access to a specific resource. Most of the processing implicated to return a service usually occurs on the server, leaving the client processing relatively low. Most popular Internet applications including the World Wide Web, FTP, telnet and email use this service-delivery model.

Client/server enabled an unsophisticated client to obtain information. All the user had to do to access the information was simply point and click. The browsers hid complex addresses and network peculiarities from the user's view. Although the Web, as it became known, enabled simple client access, it did not enable simple publishing. Whereas a consumer need only aster a mouse, a publisher had to master HTML, DNS registration, web servers, Common Gateway Interface (CGI), and a litany of other acronyms and buzzwords. This shifted the original P2P balance between consumers and acronyms and buzzwords. This shifted the original P2P balance between consumers and producers: Big organizations had the time and skills to master the production complexity, while individual users became pure consumers.

Almost unnoticed, the Internet continued to evolve. Although, people loved the browser, and were often impressed by the graphic glitz of HTML pages, the Internet capability they depended on most was email, which many cite as the seminal P2P application. Email as simple text exchange became the Internet's killer application. Email achieves P2P goals with some limitations, by delivering information from one person to another. It perfectly balances easy production and easy consumptions. As studies show, e-mail is the one Internet function that would be difficult for users to give up.

Unfortunately, this architecture has a major drawback. As the number of clients increases, the load and bandwidth demands on the server also increase,eventually preventing the server from handling additional clients. The advantage of this architecture is that it requires less computational power on the client side.

The client in the client/server architecture acts in a passive role, capable of demanding services from servers but incapable of providing services to other clients. This model of service delivery was developed at a time when most machines on the Internet had a resolvable static IP address.

Peer-to-peer technology enables any network-aware device to provide services to another network aware device. A device in a P2P network can provide access to any type of resource that it has at its disposal, whether documents, storage capacity, computing power, or even its own human operator. The technology is a natural extension of the Internet's philosophy of robustness through decentralization. In the same manner that the Internet provides domain name lookup (DNS), World Wide Web,email,and other services by spreading responsibility among millions of servers,P2P has the capacity to power a whole new set of robust applications by leveraging resources spread across all corners of the Internet.

As the Internet grew, the finite supply of IP addresses prompted service providers to begin dynamically allocating IP addresses to machines each time they connected to the network through dial-up connections. The dynamic nature of these machines' IP addresses effectively prevented users from running useful servers. Although someone could still run a server, that user couldn't access it unless he knew the machine's IP address beforehand. These computers form the "edge" of the Internet: machines that are connected but incapable of easily participating in the exchange of services. For this reason,most useful services are centralized on servers with resolvable IP addresses,where they can be reached by anyone who knows the server's easy-to-remember domain name.

4.2.2 Elements of P2P Networks

To implement a peer-to-peer network first it's necessary to satisfy some basic questions about information, resources and services [19]:

- Locate other devices.
- Organize to address common interests.
- Inform about available capabilities.
- Naming system to identify a device in a unique way.
- Exchange messages and data.

All P2P networks are built on some fundamental elements to satisfy these necessities and others. Unfortunately, many of these elements are assumed or implied by proprietary P2P networks and are hard-coded into many P2P applications' implementations, resulting in inflexibility. For instance, the majority of current P2P solutions assumes the use of TCP as a network transport mechanism and cannot operate in any other network environment. Flexible P2P solutions need a language that explicitly declares all of the variables in any P2P solution. The following sections define the basic terminology of P2P networking.

In the following we describe the elements and components to achieve a P2P communication using the JXTA framework [28] [26].

Peers

A peer is a node on a P2P network that forms the fundamental processing unit of any P2P solution. A peer can be represented by an instance of a distributed application running on different machines, or a peer can be a portable device that runs a small application. It is a very flexible definition that it's not tied to an specific situation. Generally a peer is described as a single machine or device, but it's not always true, a machine can be running several peer instances.

Any entity capable of performing some useful work and communicating the results of that work to another entity over a network, either directly or indirectly.

The definition of useful work depends on the type of peer. Three possible types of peers exist in any P2P network:

- Simple peers: A simple peer is designed to serve a single end user, allowing that user to provide services from his device and consuming services provided by other peers on the network. In all likelihood, a simple peer on a network will be located behind a firewall, separated from the network at large; peers outside the firewall will probably not be capable of directly communicating with the simple peer located inside the firewall.

  Because of their limited network accessibility, simple peers have the least amount of responsibility in any P2P network. Unlike other peer types, they are not responsible for handling communication on behalf of other peers or serving third-party information for consumption by other peers.

- Rendezvous peers: Taken literally, a rendezvous is a gathering or meeting place; in P2P, a rendezvous peer provides peers with a network location to use to discover other peers and peer resources. Peers issue discovery queries to a

rendezvous peer, and the rendezvous provides information on the peers it is aware of on the network.

A rendezvous peer can augment its capabilities by caching information on peers for future use or by forwarding discovery requests to other rendezvous peers. These schemes have the potential to improve responsiveness, reduce network traffic,and provide better service to simple peers.

A rendezvous peer will usually exist outside a private internal network's firewall. A rendezvous could exist behind the firewall, but it would need to be capable of traversing the firewall using either a protocol authorized by the firewall or a router peer outside the firewall.

- Router peers: A router peer provides a mechanism for peers to communicate with other peers separated from the network by firewall or Network Address Translation (NAT) equipment. A router peer provides a go-between that peers outside the firewall can use to communicate with a peer behind the firewall, and the opposite.

  To send a message to a peer via a router, the peer sending the message must first determine which router peer to use to communicate with the destination peer. This routing information provides a mechanism in P2P to replace traditional DNS, enabling an intermittently connected device with a dynamic IP address to be found on the network. In a similar manner to the way that DNS translates a simple name to an IP address, routing information provides a mapping between a unique identifier specifying a remote peer on the network and a representation that can be used to contact the remote peer via a router peer.

  In simple systems,r outing information might consist solely of resolving an IP address and a TCP port for a given unique identifier. A more complex system might provide routing information consisting of an ordered list of router peers to use to properly route a message to a peer. Routing a message through multiple router peers might be necessary to allow two peers to communicate by using a router peer to translate between two different and incompatible network transports.

Peer Groups

Before JXTA, the proprietary and specialized nature of P2P solutions and their associated protocols divided the usage of the network space according to the application Generally to perform file sharing, you probably used the Gnutella protocol

and could communicate only with other peers using the Gnutella protocol; similarly, if you wanted to perform instant messaging, you used ICQ and could communicate only with other peers also using ICQ.

The protocols' incompatibilities of current P2P applications like GNutella or BitTorrent or IM, effectively divide the network space based on the application being used by the peers involved. Considering a P2P system in which all clients can speak the same set of protocols, the concept of a peer group is necessary to subdivide the network space. A peer group is defined as follows:

A set of peers formed to serve a common interest or goal dictated by the peers involved. Peer groups can provide services to their member peers that aren't accessible by other peers in the P2P network. Peer groups divide the P2P network into groups of peers with common goals based on the following:

- The application they want to collaborate on as a group. A peer group is formed to exchange service that the members do not want to have available to the entire population of the P2P network. One reason for doing this could be the private nature of the data used by the application.
- The security requirements of the peers involved. A peer group can employ authentication services to restrict who can join the group and access the services offered by the group.
- The need for status information on members of the group. Members of a peer group can monitor other members. Status information might be used to maintain a minimum level of service for the peer group's application.

Network Transport

To exchange data, peers must employ some type of mechanism to handle the transmission of data over the network. This layer, called the network transport, is responsible for all aspects of data transmission, including breaking the data into manageable packets, adding appropriate headers to a packet to control its destination, and in some cases, ensuring that a packet arrives at its destination. A network transport could be a low-level transport, such as UDP or TCP, or a high-level transport, such as HTTP or SMTP.

The concept of a network transport in P2P can be broken into three constituent parts:

- Endpoints: The initial source or final destination of any piece of data being transmitted over the network. An endpoint corresponds to the network interfaces used to send and receive data.

- Pipes: Unidirectional, bidirectional, asynchronous, virtual communications channels connecting two or more endpoints.

- Messages: Containers for data being transmitted over a pipe from one endpoint to another. To communicate using a pipe, a peer first needs to find the endpoints, one for the source of the message and one for each destination of the message, and connect them by binding a pipe to each of the endpoints. When bound this way, the endpoint acting as a data source is called an output pipe and the endpoint acting as a data sink is called an input pipe. The pipe itself isn't responsible for actually carrying data between the endpoints; it's merely an abstraction used to represent the fact that two endpoints are connected. The endpoints themselves provide the access to the underlying network interface used for transmitting and receiving data.

To send data from one peer to another, a peer packages the data to be transmitted into a message and sends the message using an output pipe; on the opposite end a peer receives a message from an input pipe and extracts the transmitted data.

Notice that a pipe provides communication in only one direction, thus requiring two pipes to achieve two-way communication between two peers. The definition of a pipe is structured this way to capture the lowest common denominator possible in network communications, to avoid excluding any possible network transports. Although bidirectional communication is the norm in modern networks, there's no reason to exclude the possibility of a unidirectional communications channel in the definition because any bidirectional network transport can easily be modeled using two unidirectional pipes.

Services

Services provide functionality that peers can engage to perform "useful work" on a remote peer. This work might include transferring a file, providing status information, performing a calculation, or basically doing anything that you might want a peer in a P2P network to be capable of doing. Services are the motivation for gathering devices into a P2P network; without services, there is not a P2P network, it's like having a set of devices incapable of leveraging each other's resources.

Services can be divided into two categories:

- Peer services: Functionality offered bya particular peer on the network to other peers. The capabilities of this service will be unique to the peer and will be available only when the peer is connected to the network. When the peer disconnects from the network, the service is no longer available.

- Peer group services: Functionality offered by a peer group to members of the peer group. This functionality could be provided by several members of the peer group, thereby providing redundant access to the service. As long as one member of the peer group is connected to the network and is providing the service, the service is available to the peer group.

Most of the functionality required to create and maintain a P2P network, such as the underlying protocols required to find peers and resources, could also be considered services. These core services provide the basic P2P foundation used to build other, more complex services.

Advertisements

Until now, P2P applications have used an informal form of advertisements. In Gnutella, the results returned by a search query could be considered an advertisement that specifies the location of a specific song file on the Gnutella network. These primitive advertisements are extremely limited in their purpose and application. At its core, an advertisement is defined as follows:

A structured representation of an entity, service, or resource made available by a peer or peer group as a part of a P2P network.

Protocols

A way of structuring the exchange of information between two or more parties using rules that have previously been agreed upon by all parties. In P2P, protocols are needed to define every type of interaction that a peer can perform as part of the P2P network:

- Finding peers on the network.

- Finding what services a peer provides.

- Obtaining status information from a peer.

- Invoking a service on a peer.

- Creating, joining, and leaving peer groups.

- Creating data connections to peers.
- Routing messages for other peers.

The organization of information into advertisements simplifies the protocols required to make P2P work. The advertisements themselves dictate the structure and representation of the data, simplifying the definition of a protocol. Rather than passing back and forth raw data, protocols simply organize the exchange of advertisements containing the required information to perform some arbitrary functionality.

Entity Naming

Most items on a P2P network need some piece of information that uniquely identifies them on the network:

- Peers: A peer needs an identifier that other peers can use to locate or specify it on the network. Identifying a particular peer could be necessary to allow a message to be routed through a third party to the correct peer.
- Peer groups: A peer needs some way to identify which peer group it would like to use to perform some action. Actions could include joining, querying, or leaving a peer group.
- Pipes: To permit communication, a peer needs some way of identifying a pipe that connects endpoints on the network.
- Contents: A piece of content needs to be uniquely identifiable to enable peers to mirror content across the network,thereby providing redundant access. Peers can then use this unique identifier to find the content on any peer.

In traditional P2P networks, some of these identifiers might have used network transport-specific details; for example,a peer could be identified by its IP address. However, using system-dependent representations is inflexible and can't provide a system of identification that is independent of the operating system or network transport. In the ideal P2P network, any device should be capable of participating, regardless of its operating system or network transport.

4.2.3 P2P Communication

The fundamental problem in P2P is how to enable the exchange of services between networked devices. To achieve the exchange it's necessary to solve two issues:

- Find peers and services on a P2P network.

- Enable peers that belong to a private network to participate in a P2P.

The most important topic is the first one, without the knowledge of the existence of a peer or a service on the network, there's no possibility for a device to engage that service. The second issue has an increasing relevance nowadays that many devices are separated from the network by networking devices designed to restrict direct connections between different devices in different networks.

Finding Advertisements

Any of the elements stated previously can be represented as an advertisement, and that characteristic considerably simplifies the problem of finding peers, peer groups, services, pipes, and endpoints. With the advertisement abstraction the main objective now is to find advertisement that can represent any element.

A peer can discover an advertisement in three ways:

- No discovery.
- Direct discovery.
- Indirect discovery.

The first technique involves no network connectivity and can be considered a passive discovery technique. The other two techniques involve connecting to the network to perform discovery and are considered active discovery techniques.

Discovering Rendezvous and Routing Peers

For most peers existing on a private internal network,finding rendezvous and router peers is critical to participating in the P2P network. Because of the restrictions of a private network's firewall, a peer on an internal network has no capability to use direct discovery to perform discovery outside the internal network. However, a peer might still be capable of performing indirect discovery using rendezvous and router peers on the internal network.

In most P2P applications, the easiest way to ensure that a simple peer can find rendezvous and router peers is to seed the peer with a hard-coded set of rendezvous and router peers. These rendezvous and router peers usually exist at static, resolvable IP addresses and are used by a peer as an entrance point to the P2P network. A peer located behind a firewall can use these static rendezvous peers as a starting point for discovering other peers and services and can connect to other peers using the static set of router peers to traverse firewalls.

4.2.4 Challenges to Direct Communication

The use of firewalls and NAT by corporate private networks poses a serious obstacle to P2P networking. NAT and firewalls are usually used together to secure a corporate network against unauthorized network activity originating from either inside or outside the network and to provide a private internal networking environment.

Firewalls

Firewalls are used to protect corporate networks from unauthorized network connections, either incoming from the outside network or outgoing from the internal network. Typically firewalls use IP filtering to regulate which protocols may be used to connect from outside the firewall to the internal network or vice versa. A firewall might also regulate the ports used by outside clients to initiate inbound connections to the internal network or by internal clients to initiate outbound connections from the internal network.

Because a firewall might block incoming connections, a peer outside the firewall will most likely not be capable of connecting directly to a peer inside the firewall. A peer within the network might also be restricted to using only certain protocols (such as HTTP) to connect to locations outside the firewall, further limiting the types of P2P communication possible.

Network Address Translation (NAT)

NAT is a technique used to map a set of private IP addresses within an internal network to another set of external IP addresses on a public network. NAT comes in two varieties:

- Static NAT: In static NAT the mapping relationship between internal and external IP addresses is one-to-one. Every internal IP address is mapped to one and only one external IP address.
- Dynamic NAT: maps the set of internal IP addresses to a smaller set of external IP addresses. A private network employing NAT usually assigns internal IP addresses from one of the ranges of IP addresses defined specifically for private networks:
  - Class A private addresses: 10.0.0.0 through 10.255.255.255
  - Class B private addresses: 172.16.0.0 through 172.31.255.255
  - Class C private addresses: 192.168.0.0 through 192.168.255.255

A machine using an IP address within this range is most likely behind NAT equipment. NAT is used for a variety of reasons, the most popular reason being that it eliminates the need for global unique IP addresses for every workstation within a corporation, thereby reducing the cost of a corporate network. NAT also enables system administrators to protect a network by providing only a single point of entry into the internal network. NAT accomplishes this by allowing only incoming connections to internal machines that originally initiated a connection to the outside network. Rather than attempting to protect each machine using a firewall to filter incoming connections, a system administrator can use NAT to ensure that the only connections allowed back into the network are those that originated within the network.

NAT is usually implemented by a router or a firewall acting as a gateway to the Internet for the private internal network. To map a packet from an internal IP address to an external IP address, the router does the following:

- Stores the source IP address and port number of the packet in the router's translation table.
- Replaces the source IP address for the packet with one of the IP addresses from the router's pool of public IP addresses, storing the mapping of the original IP address to the public IP address in the translation table in the process.
- Replaces the source port number with a new port number that it assigns and stores the mapping in the translation table. After each step has been performed, the packet is forwarded to the external network. Data packets arriving at one of the router's external public IP addresses go through an inverse mapping process that uses the router's translation table to map the external port number and IP address to an internal IP address and port number. If no matching entry for a given public IP address and port number is found in the translation table, the router blocks the data from entering the internal private network.

NAT protects networks by allowing only connections to the internal network that originated within the internal network. A machine outside the network can't connect to a machine in the internal network unless the internal machine initiated the connection to the external machine. As a result, an external peer in a P2P network has no mechanism to spontaneously connect to a peer located behind a NAT gateway. From the outside peer's point of view, the peer doesn't exist because no mapping between external and internal IP addresses and port numbers exists in the router's translation table.

4.2.5 Routing Messages Between Peers

When a firewall or NAT is located between two peers, a router peer must be used to proxy a connection between the public network and the peer located inside the firewall. In the simple case, only a single firewall separates the source and destination peers, thus requiring only a single router peer. In more complex cases, a firewall or NAT can protect each of the peers and require the use of multiple router peers to traverse each firewall/NAT boundary.

To allow a peer located inside a firewall/NAT to send a message to another peer located on the public network, three steps are required:

- The peer behind the firewall/NAT connects to the router peer using a protocol capable of traversing the firewall, such as HTTP, and requests that the router peer forward a message to a destination peer.
- The router accepts the connection from the peer behind the firewall and initiates a connection to the requested destination on the peer's behalf. This connection uses whatever network transport both the router peer and the destination peer have in common.
- The message is sent from the source to the destination peer by the router peer,acting as a proxy for the source peer.

After the message from the source peer has been sent to the destination peer, the connection closes. Further messages can be sent by repeating the procedure, but the message might use a different router peer and, therefore, might follow a different route to the destination peer.

To allow a public peer to send a message to a peer located behind a firewall/-NAT, the source peer must know routing information that describes a router peer capable of routing the message to the destination peer. Route information might have been obtained previously during discovery or might require an additional discovery request to the P2P network. When the source peer has obtained routing information, sending the message involves three steps:

- The source peer opens a connection to the router peer asking it to forward the message on to the destination peer.
- The router peer waits until the destination peer connects to it using a protocol capable of traversing the firewall.
- The destination peer connects to the router peer periodically, at which point the message is pushed down to the destination peer.

Again, when the message reaches the destination peer, the connection between the router peer and the other two peers is closed. Sending another message from the source peer requires repeating the procedure and might use a different router peer to provide connectivity to the destination peer.

4.2.6 Traversing the NAT/Firewall Boundary

The combined use of NAT and firewalls results in an especially difficult set of circumstances for peer communication: Peers can't connect to machines behind NAT unless the internal peer initiates communication, and connections can be blocked at the firewall based on the connection's protocol or destination IP address and port number.

The only tool that a peer has at its disposal to solve this problem is its capability to create outgoing network connections to hosts outside the firewall/NAT gateway. Peers can use protocols permitted by the firewall to tunnel connections through the firewall to the outside network. By initiating the connection within the internal network, the necessary mapping in the NAT router translation tables is set up, allowing an external machine to send data back into the internal network. However, if a firewall is configured to deny all outgoing connections, peer communication is impossible.

In most corporate networks, HTTP is the protocol most likely to be enabled by a firewall for outgoing connections. Unfortunately, HTTP is a request-response protocol: each HTTP connection sends a request and then expects a response. The connection must remain open after the initial request to receive the response. Although HTTP provides a peer with a mechanism to send requests out of the internal network, it doesn't provide the capability for external peers to spontaneously cross the firewall boundary to connect to peers inside the internal network. To address this problem, a peer inside a firewall uses a router peer either located outside the firewall or visible outside the firewall to traverse the firewall. Peers attempting to contact a peer behind a firewall connect to the router peer, and the peer behind the firewall periodically connects to a router peer. When the internal peer connects to the router, any incoming messages get pushed down to the peer in the HTTP response.

This technique can be used with any protocol permitted by the firewall and understood by the router peer. The router peer effectively translates between the network transport used for P2P communication and the transport used to tunnel through the firewall.

4.2.7 Double Firewall/NAT Traversal

Most simple peers located at the edgeof the Internet are likely to be protected by a firewall/NAT,so any message being sent from a source peer to a destination peer will need to traverse two firewall/NAT boundaries. The procedure for traversing two firewalls is similar to the single firewall traversal case and basically combines both the incoming and the outgoing cases of the single firewall traversal scenario.

Before a source peer can send the message, it needs to locate routing information for the peer that describes a set of router peers capable of proxying messages to the destination peer. In this case, more than one router peer might be involved; one router peer is required to allow the source peer to traverse its firewall, and another is required to traverse the firewall providing access to the destination peer. When the source peer has this routing information, sending the message involves four steps:

- The source peer opens a connection to the source router peer,asking it to forward the message on to the destination peer by way of the destination router peer provided.
- The source router peer opens a connection to the destination router peer. This connection uses whatever network transport both router peers have in common.
- The destination router peer waits until the destination peer connects to it using a protocol capable of traversing the firewall,such as HTTP.
- The destination peer connects to the router peer periodically,and the message is pushed down to the destination peer. Traversing both firewalls might involve only one router peer if both the source and the destination peers have a router peer in common. However, traversing firewall boundaries isn't the only reason to use a router peer. Multiple router peers can be used by a peer to circumnavigate network bottlenecks and achieve greater performance, or to provide translation between two incompatible networks transports. When the peer connects to the source router peer in this case, it provides an ordered list of router peers to use to send the message to the peer on its behalf.

4.3 Neubot Peer-to-Peer Services

Based on the JXTA protocols and services, we designed an architecture for the system trying to employ the maximum capacity of each service implemented in the JXSE binding.

In the "bootstrap" step every peer gets connected to the standard NetPeerGroup defined by the JXTA community. This peer group is hardcoded and forms part of the bootstrap process of the platform. After the peer has joined the group, it has to connect to the POLITO rendezvous, this is done to verify the capacity of the peer to reach Internet and in consequence the rendezvous peer, after it has connected it can communicate with the rest of the world.

The second step after obtaining a release from the rendezvous is to create and join the NEUBOT peer group. In this first implementation the NEUBOT group serves only as a scope group, because it is not implementing any secure membership service in order to give an identity to the peers with a secure mechanism, but it allows in the future to develop a secure group with a membership service able to control the peers that can access the group. At the moment all peers can join this group. After joining the group the peer waits for a rendezvous that participates inside the Neubot peer group (POLITO rendezvous) and at this point the peer is able to send messages to all the peers registered with the rendezvous. The explanation for this aspect is simple: the endpoint protocol of JXTA uses all the transport protocols available in the peer endpoint, when using JXTA over an IP network the first transport protocol used is IP multicast sending messages to a multicast IP family like 225.0.0.1. In our specific case the peers are widespread over Internet and this transport protocol is useless. In consequence JXTA uses the TCP or HTTP transport protocol to propagate the message and by default it sends the message to the Rendezvous peer and all its known peers, and it's the Endpoint Routing Protocol in charge of delivering the message to the right peer.

4.3.1 Presence Service

After the peers are connected to the rendezvous they can receive all JXTA messages. The next step is to start the presence service, although the name, it is not a traditional presence service where a peer is able to have a list of other peers in the network. The idea is to announce that the peer is looking another peer to do the measurements (searching message), if nobody is online the peer stays online listening for presence messages.

The peers can be in three basic states:

- Idle: it means that it is waiting for another peer to do the measurements.
- Paired: that it found another peer and they are negotiating the protocol to measure.

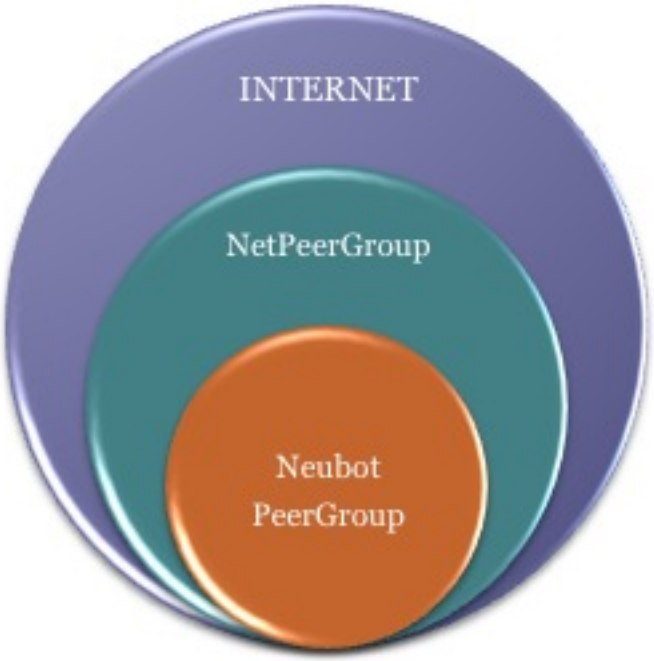

Figure 4.1. Peer Group Situation

- Measuring: they are exchanging data with other peer in an specific traffic protocol (BitTorrent, SIP, others)

The simplicity of the presence service allows to find a free peer in the moment the request is sent. As this is not a traditional P2P application that intensely needs a prolonged connection, to exchange files, and the application does not need to contact a busy peer, contact a peer while the requested peer is in contact with other peer.

To implement the described presence service it is required to follow the next stages:

The first stage is to create an output propagated pipe and send a searching message through this pipe; the ID of the propagated pipe is hardcoded into the application, in order to connect all peers to the same pipe. If the peer sends the message and receives no answer, after a default time-out, it means that is the only one looking for another peer to do the measurements. In consequence it binds an input propagated pipe with the same hard-coded pipe ID in order to receive searching messages from other peers. When a new peer enters the network it sends the searching message and they can complete the objectives. The first

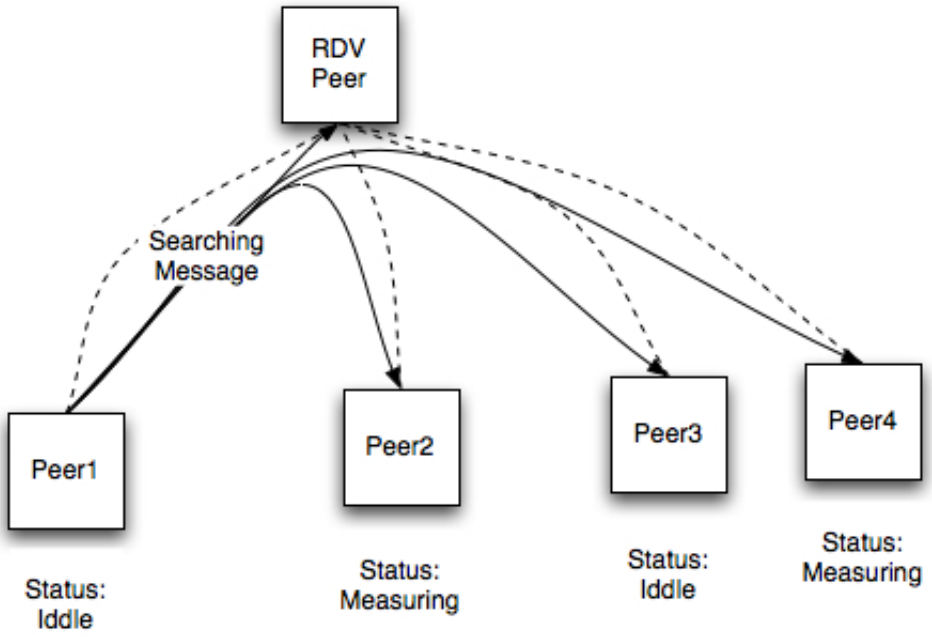

Figure 4.2. Presence Message Exchange

and unique searching message, that a peer sends through the propagated pipe, contains a BiDiPipe advertisement to establish a bidirectional communication channel with the other peer. After the peer sends the searching message it opens a Server Pipe to wait for another peer to connect to the BiDiPipe, if some peer receives the message and it's available to perform the measures it connects to the Server Pipe to create a BiDiPipe with the other peer. After this moment we can state that the presence service accomplished the target, finding an available peer to perform the network measures passing to the paired state. Now it is time to instantiate the Neutrality service in charge of doing the network measurements. When the peers finish the measurement it returns again to the idle state.

In case the searching message sent is not answered the peer creates an input BiDiPipe to listen for new incoming searching messages, this messages can come from new peers joining the network or can be sent by peers that have finished a measure to contribute with other peers. It is essential to mention that the peer while waiting the resources consumption is reduced to minimum because the exchanged messages is null, it just listens to incoming messages.

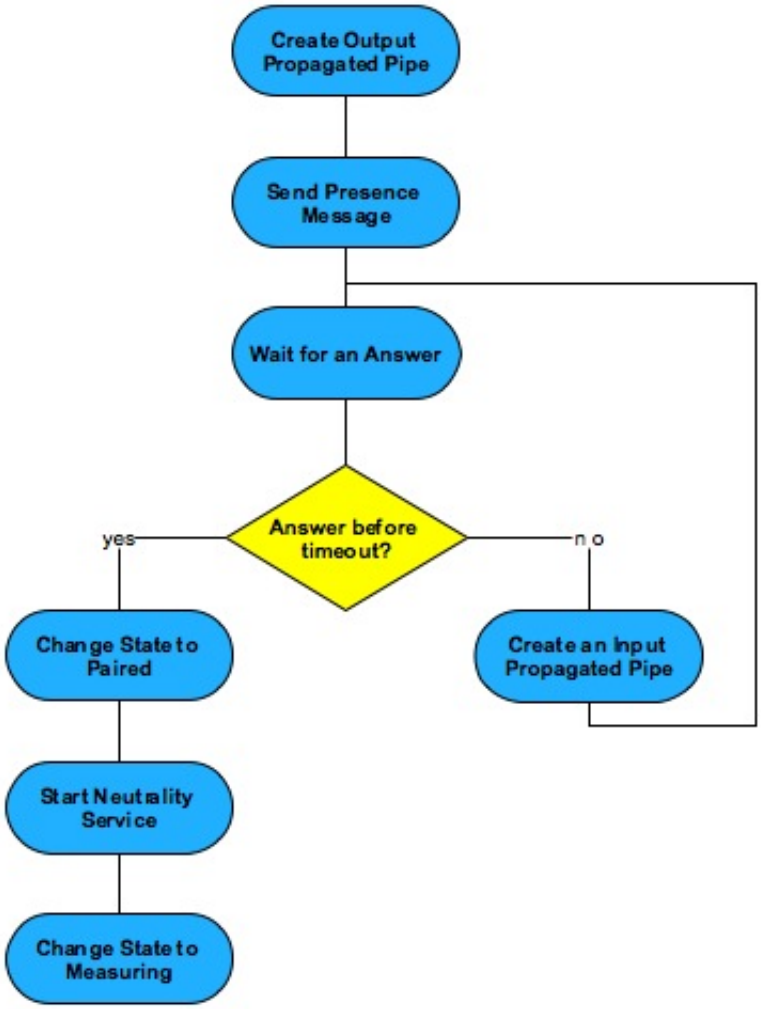

Figure 4.3. Presence Algorithm

4.3.2 Neutrality Service

The Neutrality service has the four main tasks that can be considered as sub-services or wrapped services inside this main service:

- Traffic Generator: The traffic generator has the main task to exchange the main parameters between both peers to establish a specified protocol or service connection, execute some basic task of the protocol and then return, in order to obtain some basic information about the behavior of the protocol in both ends of the connection.

In order to start the protocol between both peers it is indispensable to exchange the parameters respecting to the protocol requested. This operation is done by sending messages through a BiDiPipe connecting both peers, this pipe is established thanks to the presence service as explained before.

When the requesting connection to the Server Pipe is accomplished, it initiates the traffic generator protocol message to establish the different parameters between the peers. First, the peer that requested the connection to the Server Pipe sends automatically a request message asking the other peer to execute a measure test

- Captures Exchange: After the capture is finished both peers needs to exchange the captured files, this in order to compare them.

  After the presence service establishes the BiDiPipe between both peers the Neutrality service is instantiated, first it is initiated the traffic generator to establish the protocol and the role of each peer in that protocol.

- Traffic capturer: It merely consists of a capturing packet loop, that captures packets being exchanged between the two peers by filtering packets based on both IP addresses. Although this packet capturer does not limit only to capture the traffic generated by the protocol, in any case the relevant thing is to capture the packets between both peers. The traffic capturer starts just before the protocol module itself, that is to say the traffic generator.

  Using JPCAP library this service needs to begin to capture the traffic in both peers, it has to start before the traffic generator begins to work and needs to stop when the traffic generator has finished to exchange data. Also this service needs to be synchronized with the exchange of JXTA messages.

  Implementing the capturer to work in almost all platforms was not an easy task, although I used a multi-platform library able to capture packets in Windows, Linux and Mac OS X, the behavior of the library is far from equal in all of them. For instance the loop captures using timeouts worked well in some OS while in other the timeout option is not working. So, to obtain the same capture in all platforms the capture loop is implemented using the basic functionalities capturing packet by packet in a loop. Storing the packets in the file was also a problematic task, to solve this problem we decided to store the packets in using Java facilities to store objects in files. This implied a lack of standard storage because we are not using pcap standard to store the file.

To synchronize and start the modules described previously Neubot defines a protocol based on query/response messages to start each module. Each protocol measure is treated as a transaction, a transaction can be defined as the process to negotiate, start and complete a protocol measure, it has an ID and the working sequence is the following:

- Request message: this message contains the protocol to measure an has different fields like country and providers that are not currently implemented, included in the design as future options.

```
Element: NeubotPresence :: MessageType [request]
Element: NeubotPresence :: TransactionCode [123]
Element: NeubotPresence :: Country [Italy]
Element: NeubotPresence :: Provider [-]
Element: NeubotPresence :: Protocol [Bittorrent]
Element: NeubotPresence :: Available [true]
```

Figure 4.4. Message: Protocol Request

- Response message: it acknowledges about using a protocol if it is available.

```
Element: NeubotPresence :: MessageType [response]
Element: NeubotPresence :: TransactionCode [123]
Element: NeubotPresence :: Country [Italy]
Element: NeubotPresence :: Provider [-]
Element: NeubotPresence :: Protocol [Bittorrent]
Element: NeubotPresence :: Available [true]
```

Figure 4.5. Message: Protocol Response

- Protocol Initialization: When the protocol has been initialized the peer that received the request send the role it is going to play in the protocol (i.e. BitTorrent the requested peer is doing as tracker).

- Protocol Started: After the requested peer has finished of starting the protocol it sends the necessary parameters to start the protocol to the other peer (i.e. the tracker URL), after this message both peers start the traffic capturer module.

```
Element: NeubotPresence :: MessageType [ProtocolInitialization]
Element: NeubotPresence :: Protocol [Bittorrent]
Element: NeubotPresence :: TransactionCode [123]
Element: NeubotPresence :: Role [Tracker]
```

Figure 4.6. Message: Protocol Initialization

```
Element: NeubotPresence :: MessageType [ProtocolStarted]
Element: NeubotPresence :: Protocol [Bittorrent]
Element: NeubotPresence :: Transaction [123]
Element: NeubotPresence :: TrackerURL [http://x.x.x.x:12345/metainfo.torrent]
```

Figure 4.7. Message: Protocol Started

- Protocol Finishing: When one of the peers detects the finishing of the protocol, due to downloaded finished or time-out, it sends this message to indicate the other peer to finish the protocol session.
- Protocol Finished: It is sent by the peer that received the finishing message to indicate that it finished the protocol and can stop the capture.

The sequence is executed with the path described previously. If the other peer has the protocol implementation available it answers with a positive response message. And starts loading the protocol, in this first version of Neubot I have implemented the BitTorrent protocol using a GNU library from a project called The Hunting of the Snark Project [16]. In the specific case of BitTorrent the peer starts a tracker with a predefined file generates the metadata file (.torrent) and publishes it, then sends to the other peer the URL of the metadata file. At this point the peer that receives the URL starts a lightweight BitTorrent client to download the file.

The situation with the BitTorrent protocol we are measuring is particularly interesting because Neubot is able to verify if the peers are able to establish a tracker, connect to it and download/upload a file from a tracker. Also using the traffic capturer module in the Neutrality Service Neubot is able to determine

```
Element: NeubotPresence :: MessageType [ProtocolFinishing]
Element: NeubotPresence :: Protocol [BitTorrent]
Element: NeubotPresence :: Transaction [123]
```

Figure 4.8. Message: Protocol Finishing

Element: NeubotPresence :: MessageType [ProtocolFinished]
Element: NeubotPresence :: Protocol [BitTorrent]
Element: NeubotPresence :: Transaction [123]

Figure 4.9. Message: Protocol Finished

which packets were lost in the BitTorrent communication, this can give a valuable information because it can be that the ISP can be throttling traffic or doing some packet spoofing to block TCP connections. However, with the results we are not able to determine wether the packet was lost intentionally by some ISP in the middle or if it was simply a lost packet in the network due to congestion, but if the number of packets is considerably high or the download/upload of the file through BitTorrent can't be completed, it is necessary to investigate the situation to see if it corresponds to an ISP problem.

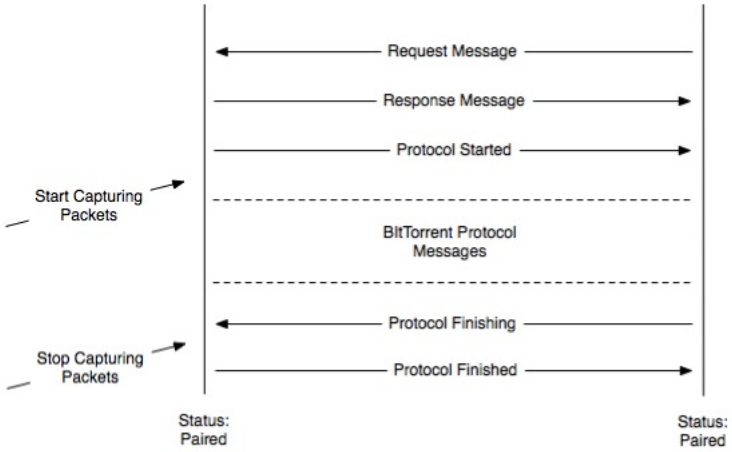

Figure 4.10. Message Flow, Neutrality Service

4.4 Storing the Results

As the client/server part and the P2P part are independent and each of them uploads the results after finishing the test, these results are stored in a database composed of two tables one for each test:

- SpeedTest: from the client/server the results correspond to some characteristics of the broadband connection, the throughputs are expressed in kbps and

time in milliseconds.

  - IP address.
  - TCP BTC Download and Upload speed in kbps.
  - UDP BTC Download and Upload speed in kbps.
  - RTT msec.
  - Date and time.

- BitTorrent: the data obtained from the BitTorrent protocol is the quantity of downloaded and uploaded data and if the download and upload was successful. With this data we are able to determine wether if the peers were able to download completely the file, if not how many data it could download to see if it is a throttling problem, or a completely blocking issue. If it was on both sides and which side is affected.

  - IP address.
  - Downloaded number of bytes.
  - Successful download.
  - Uploaded number of bytes.
  - Successful upload.
  - Number of not found packets when downloading.
  - Number of not found packets when uploading.
  - Date and time.

4.5 Used Technologies

4.5.1 Java

Java is a platform from Sun Microsystems that integrates the necessary tools and specifications for developing cross-platform software. A Java program runs on the Java Runtime Environment, this JRE is available for almost all OSs (Windows, Mac OS X, Unix/Linux and even mobile platforms). Code written with Java programming language is compiled to Java bytecode; then this bytecode is executed by the Java Virtual Machine.

The Java Virtual Machine (JVM) has a Just in Time compiler that is in charge of translating the Java bytecode to native instructions at run time. The JVM is also in charge of memory management implementing an automated garbage

collection freeing the programmer of optimizing memory usage. Java is object oriented with a class library to provide most common functions and utilities necessary to do specific tasks, this library is called the Java API and is as much a part of Java than the language itself.

Advantages

- Platform independence, using the Virtual Machine approach Java does not need to have native compilers for each platform. Instead, the compiler translates Java into the machine language of the Java Virtual Machine, which is called bytecode. Then the JVM executes the bytecode.
- Java has been the Internet programming language having very good support for Internet applications.
- Large library support having a good API for most common operations.

Disadvantages

- Performance can be considered a disadvantage due to the JVM approach of the language. However, the system is mainly based on network operations which are slower than the execution process.
- Java lacks support for low level operations necessary for the development of Neubot like raw access to Network Interfaces being an obstacle to overcome by using a third party library.
- Based on the selection of Java as a multi platform language that ease the portability of the project between OSs, a plausible and probably unique solution to access the network interface in Java is JPCAP.

4.5.2 Jpcap

JPCAP is a Java library for capturing and sending network packets. It is a multi platform library successfully tested in the following OS:

- Windows XP: Installed and tested a sample application to capture and send packets, verified and compared with Wireshark. Only Wifi interface.
- Linux dist. Debian etch: Installed and tested with a sample application to capture and send packets, verified and compared with Wireshark. Only Wifi interface.

- Mac OS X ver. 10.4: Installed and compiled source of sample application. Running using the Wifi interface the OS seems to have the same problem as Wireshark, it disconnects the interface when listing the network interfaces. Todo: test selecting directly the Wifi interface and with the other interfaces (the way wireshark solves the problem).

Advantages

- Language integration with Java for capturing network packets, otherwise it would be necessary to develop C language libraries using Libpcap for Linux/Unix systems and Winpcap for Windows systems.
- Language integration with Java to send raw packets through the network interface, otherwise the usage of raw sockets using the OS APIs is mandatory to open raw sockets.
- In addition having the library for several platforms allows to develop a module that works with the network interface directly instead of developing one module for each OS.

Disadvantages

- The behavior of the library varies having slight changes from OS, parameters like time-outs or promiscuous mode can or cannot work in some OS. As usual the support for Windows is complete so a strategy to take into account to overcome this problem is to develop in a Linux/Unix environment testing after having a significant change or progress the project on the other platform to adjust an incongruence that may occur.
- The necessary requirements to use JPCAP, specifically the version in Linux platforms. For the tested case (Debian etch) it was necessary to install the requirements using the package administrator, if a requirement is installed without using the package facility it must be registered as an installed package because the installer verifies the package and not that the requirement itself was installed. To install JPCAP in Windows there were no problems but it is necessary a previous installation of the requirements. To solve the problem of the requirements it is recommended to make an installer capable of solving all the requirement problems, to avoid the user dealing with this kind of problems.
- Having the possibility of sending raw packets has its disadvantages. It is necessary to fill all the packet with the correct information for all the protocols used, since the Ethernet header parameters like Mac Addresses, until the TCP parameters. This needs from the programmer an important knowledge of the

protocols used to follow the protocol RFC in order for the communication to work. In our case we are planning to use JPCAP to send simple ICMP or UDP packets, otherwise we should use the socket support offered by Java.

4.5.3 JXTA-JXSE

JXTA is a set of open P2P protocols that allow any networked device to communicate and collaborate mutually as peers. The JXTA protocols are programming language independent, and multiple implementations, also known as bindings, exist for different environments. The JXTA binding for the Java platform is suggested to be used in conjunction with the selection of the Java language for the system.

JXTA defines a series of XML messages, or protocols, for communication between peers. Peers use these protocols to discover one another, advertise and discover network resources, and communication and route messages.

Advantages

- Using JXTA-JXSE allowed the design and implementation of a lightweight protocol instead of adapting an implemented P2P protocol that is designed for other purpose like file transfer or instant Messaging (IM), that probably does not fit exactly our needs for the Neubot, because it is not a traditional P2P application, that does not need all the features implemented by general P2P protocols.
- Using a P2P architecture NEUBOT has the flexibility of not depending on different servers distributed abroad.
- Flexibility to add more measurement modules to the system that can be implemented on the P2P network by extending a previous service.
- The framework offers the network architecture offers some basic services (core services):
  - Globally unique peer addressing (JXTA Ids) to be independent of DHCP and DNS the most common standards for addressing and name resolving used on Internet.
  - Peer discovery to discover published peers and resources.
  - Transport Communications.
  - Security Primitives.

– Using the PIP and PDP allows to obtain all the information and measurement modules available in a peer, allowing the modularity and scalability between different versions or devices capabilities of the system.

Disadvantages

– JXTA is an evolving project that does not fulfill all the specifications completely. In consequence there have been major changes between different versions adapting and changing the platform continuously. Also the documentation and support due to the continuous changes in the project can be outdated or incomplete.

– The necessity of understanding and developing services and applications for the underlying protocols proposed by the JXTA standard. Probably using a client/server architecture the developing process can be simpler.

4.5.4 MySQL

MySQL is a an open source relational database management system. The program runs as a server providing multi-user access to a number of databases. A relational database stores data in separate tables rather than putting all the data in one big storeroom. This adds speed and flexibility. The SQL part of "MySQL" stands for "Structured Query Language". SQL is the most common standardized language used to access databases and is defined by the ANSI/ISO SQL Standard.

Main features:

– Internals and Portability.

– Data Types support.

– Security.

– Connectivity with different languages and platforms.

4.5.5 Java Web Start

Java Web Start is an application-deployment technology that gives you the power to launch full-featured applications with a single click from your Web browser. You can now download and launch applications, such as a complete spreadsheet program or an Internet chat client, without going through

complicated installation procedures.

Java Web Start includes the security features of the Java platform, so the integrity of your data and files is never compromised. In addition, Java Web Start technology enables you to use the latest Java SE technology with any browser.

With Java Web Start, you launch applications simply by clicking on a Web page link. If the application is not present on your computer, Java Web Start automatically downloads all necessary files. It then caches the files on your computer so the application is always ready to be relaunched anytime you wantÑeither from an icon on your desktop or from the browser link. And no matter which method you use to launch the application, the most current version of the application is always presented to you.

Java Network Launching Protocol

Java Network Launching Protocol (JNLP) is a closely-related concept that is often used interchangeably with the term "Web Start". It is the protocol, defined with an XML schema, that specifies how Java Web Start applications are launched. JNLP consists of a set of rules defining how exactly this launching mechanism should be implemented. JNLP files include information such as the location of the jar package file and the name of the main class for the application, in addition to any other parameters for the program. With a properly configured browser, JNLP files are passed to a Java Runtime Environment which in turn downloads the application onto the user's machine and starts executing it. JNLP was developed under the Java Community Process as JSR 56, which includes the original 1.0 release, the subsequent 1.5 maintenance release, and as of 2006, the pending 6.0 maintenance release. JNLP is free; developers are not required to pay a license fee in order to use it in programs.

Important Web Start features include the ability to automatically download and install a JRE in the case where the user does not have Java installed, and for programmers to specify which JRE version is needed to run a given program. The user does not have to remain connected to the Internet to execute the downloaded programs, because they execute from a locally-maintained cache. Finally, automatic updates of the software from the Web are available when the user is connected to the Internet, thus easing the burden of deployment.

Anyone can reap the benefits provided by JNLP by simply installing a JNLP Client (most commonly Java Web Start). This Client Installation can be made automatic, so that the end users can see the client launcher automatically downloaded and installed before the Java application the first time they launch the latter.

JNLP works on a classic Client-Server scheme. The JNLP Client reads and executes the JNLP File (that is the XML file) eventually contacting a JNLP Server or some web server for help. The JNLP Client runs locally on the client system whereas the server is implemented by some servlet and is used only for some advanced features of the protocol.

Advantages

- Java Web Start has a main advantage that allows to manage the application version as well as updates. Due to the JNLP protocol that connects to the server hosting the application to check the version of the application and download the adequate packages.
- A very important characteristic is the security architecture used by Java Web Start that signs the application using a Digital certificate asking the user wether or not to trust the developer of the application. But also protects the user from being sure that the application is downloading is not masqueraded by other server trying to introduce malicious code.
- Free the user from installing an application to their system, it just need to have the .jnlp file, when executed it connects to the server to download the application or verify the current version.

Disadvantages

- The ussage of Java Web Start in Linux does not let the user to execute the application as a super user, even if the user has an administrator account. In this case we need to deliver directly the executable jar because some libraries used in Neubot needs the super user privileges.
- Java Web Start still as a not very well known way to distribute applications among average users which may imply a barrier to use Neubot.

Chapter 5

Related Work: other proposals

Currently the Net Neutrality topic has become an interesting research topic for different kind of organizations, starting from Universities passing through civil associations that defends rights in the digital world, and last but not least the Internet Service Providers. In the following will be presented currently published projects and solutions that approach Net Neutrality. It is very interesting to note that each of the current projects is very different among them because each association takes a different approach to the issue depending on the possible problems perceived by the ISPs.

5.1 Electronic Frontier Foundation (EFF)

EFF is the leading civil liberties group defending rights in the digital world. The main task is to defend the freedom in our networked world by confronting cutting-edge issues defending free speech, privacy, innovation, and consumer rights today. Although their main action line is in legal cases, they have taken part in the Comcast case, an US provider that has been blocking BitTorrent uploads of their customers.

Consequently, they have analyzed and studied the Comcast case releasing two whitepapers. The first one, called Packet Forging By ISPs: A report on Comcast Affair, where they raise and explain the details of the situation and what they suspect can happen in the ISP, and individuating a possible situation of what Comcast is applying and on what kind of traffic. The second paper, Detecting packet injection: a guide to observing packet spoofing by ISPs, gives an introduction for general Internet users to explain what can be happening on ISPs and a methodology to test the ISP based on technical

tools like traffic capturers (e.g. Wireshark) and how to do basic tests with the collaboration of another Internet customer.

Continuing with the packet forging line, EFF released a software tool called pcapdiff that is able to take two captured pcap traffic traces and compare them in order to report spoofed and dropped packets between the endpoints that captured the traffic.

Although this method represents a good option, for average users is difficult to execute it because it needs the collaboration of another user in a remote machine at the same time using a traffic capturer, but it represents a valid option for more "tech" interested users in order to evaluate the ISP behavior in terms of packet spoofing.

5.2 Glasnost

The Glasnost project is developed by the Max Planck Institute for Software Systems, the goal is to make Internet access networks (DSL, cable and cellular networks) more transparent to their customers. The project is divided in two main parts:

5.2.1 Test broadband link

This first part is dedicated to measure some broadband link characteristics. The main idea of this part is to learn more about broadband services, because usually ISPs do not clearly specify the service they provide. The tool provides a method to collect data and verify the service the customer is getting. This part can be compared to the client server part of the Neubot project, actually the objective is the same. The measured parameters are the following:

- Bandwidths.
- Router queue sizes.

Both measures are done in upstream and downstream.

5.2.2 BitTorrent Traffic manipulation

As mentioned before, the creators of the Glasnost affirm that ISPs are increasingly deploying a variety of middleboxes (e.g., firewalls, traffic shapers,

censors, and redirectors) to monitor and to manipulate the performance of user applications. Most ISPs do not reveal the details of their network deployments to their customers. We believe that this knowledge is important to help users make a more informed choice of their ISP. Further, such knowledge is also useful for researchers designing protocols and systems that run on top of these networks.

With that goal in mind, the test focuses on the popular BitTorrent protocol as many ISPs are suspected to manipulate BitTorrent traffic. This type of traffic can be identified by the port it is sent on (e.g., TCP port 6881) or by BitTorrent content headers which occur in the packets.

Therefore, the designed tool is able to detect whether your ISP is using one of the following techniques:

- Throttling all BitTorrent traffic.
- Throttling all traffic at well-known BitTorrent ports.
- Throttling BitTorrent traffic only at well-known BitTorrent ports.

They have presented some results and the most important for the research community is that customers are using the tool. This means that customers are interested in knowing the service they are obtaining and becoming a collaborative part to collect the necessary data.

5.3 NNSquad Network Measurement Agent

The NNSquad Network Measurement Agent (NNMA) is developed by the Network Neutrality Squad (NNSquad) that is an Internet community with the main objective of helping to keep the Internet's operations fair and unhindered from unreasonable restrictions.

The project's focus includes detection, analysis, and incident reporting of any anticompetitive, discriminatory, or other restrictive actions on the part of Internet Service Providers (ISPs) or affiliated entities, such as the blocking or disruptive manipulation of applications, protocols, transmissions, or bandwidth; or other similar behaviors not specifically requested by their customers.

Other key aspects of the project are discussions, technology development and deployment, and associated activities aimed at keeping the Internet

a maximally unhindered, useful, competitive, fair, and open environment for the broadest possible range of applications and services.

About NNMA, it is a Windows based application (it is able to run on Windows 2000, XP, Vista). The main task is to monitor network activity on computer systems, looking for and flagging a variety of potential problems. NNMA also includes a special function that attempts to detect reset (RST) packets that may have been injected into a TCP connection by any entity not located at the connection endpoints, this is done by a method called Spoof Reset Detection Methodology. The main interest in developing the Spoof Reset Detection Methodology is the Comcast case, because that seems to be the way DPI vendors are implementing the traffic throttling.

5.3.1 Spoof Reset Detection Methodology

Packets traversing an IP network take a finite amount of time to travel between two TCP endpoints. Since all of the data sent in a TCP stream carries sequence numbers within that stream, and those sequence numbers are acknowledged ("acked") by the endpoint peer, the total time for data to be sent to the peer and for the "ack" associated with that data to return can be measured.

This round trip time (RTT) is tracked internally by TCP protocol layers, however it can also be measured by external monitoring devices or software at the endpoints. It is this methodology that is implemented by NNMA.

When sending bulk data during a TCP connection, the RTT between two TCP endpoints usually settles into a narrow, predictable range. Spoofed resets which are injected into the stream will usually have an RTT well below the measured average. Logging and analysis of these TCP "early" resets (those which show unusually low RTTs as compared against the average RTT for that given data stream) should provide a useful indication of likely spoofed reset activity.

Reset packet spoofers could attempt to evade this detection technique and improve their "stealthiness" by first measuring the RTT of a connection that they are planning to disrupt, then delaying the transmission of their spoofed reset until timing falls within the "expected" RTT. The problem with this approach is the significant risk that the spoofed reset will arrive too late

from the standpoint of the receiving endpoint.

Most TCP stacks will ignore resets that arrive after the current sequence window, but these may also potentially be useful flags that something unusual might be afoot. In short, spoofed resets have only a relatively narrow time window in which they can be both effective at disrupting connections and simultaneously be resistant to detection as potentially anomalous events. While NNMA currently only looks for early resets and not late ones, the latter is definitely an area worthy of further study.

While the reset packet detection system included in this release is of interest, NNSquad views this package as more important in the long run as a development base for a broad range of network measurement functionalities and associated communications and analysis efforts.

Chapter 6

Conclusions

Net Neutrality has become a very active topic in research, although it can be considered as a market and legislation issue. It is necessary to support the Internet community with tools to try to evaluate and determine the situation with ISPs. Thus, the design and implementation of Neubot to measure broadband connection parameters and protocol specific characteristics becomes part of the main objective of the Nexa as a Research Center for Internet & Society.

This project can be divided in two main parts, the study of Network Neutrality from a technical point of view and the design and implementation of a distributed application to measure some technical parameters about broadband connection performance and specific protocol performance. Designing and developing a distributed Internet application needs very accurate design taking into account all the needed specifications. Neubot includes a client/server architecture and a peer-to-peer architecture offering the two main approaches of today Internet in order to develop an scalable solution that is flexible and open to the developers community to be improved and extended.

Finding an approach to measuring Net Neutrality implied a very deep design part and an accurate implementation because there isn't a method to measure Net Neutrality. The idea is to measure parameters that can give an idea about the performance of the ISP, Neubot was designed to take advantage of client/server architecture to measure broadband connection parameters: download and upload throughput in TCP and UDP connections and RTT. Also the peer-to-peer architecture was used to develop an innovative method to measure protocol performance, the developed method was implemented to measure BitTorrent protocol for the first release because currently it is the

protocol causing some troubles with some ISPs in the US, so the idea was to release a first version that could be attractive for Internet users.

6.1 Future Work

The main goal of Neubot design and implementation is to be a scalable project, able to grow and expand in all its branches. With the current architecture new measures can be easily added to both main parts, client/Server and peer-to-peer.

The first expansion is to add different network protocol implementation to the P2P part to test different protocols, the work needed is to add the library to the software and add the protocol to the protocol list, and add support in the database to collect the data. In this way is easy to increase the quantity of tested protocols, the parameters to test depends on the protocol implementation.

Another important point is to add the functionality to detect a Service Provider Identification, in order to have present which ISPs where involved in each measure, to be able to detect any irregularities and analyze different situations between different providers and even between the same provider. This represents a valuable information because, some providers can affect the performance of the broadband connection when traffic is going outside the ISP network, while traffic being exchanged between customers from the same ISP, as the traffic should remain in their own network, is not altered or affected.

Some security aspects need to be designed and implemented, for instance securing communications between client/server and between peers. Currently messages are sent in clear, leaving the possibility to security threats that can affect the measures and alter the analysis done to the gathered data.

Developing a lite version of the Neubot can be a good approach to try to adapt Neubot to perform as an applet, suppressing the low level features, mainly accessing to the network interface to capture packets, to have access to even more users, that for some reason are not convinced to use Neubot.

Appendix A

JXTA-JXSE

JXTA is a set of open, generalized peer-to-peer (P2P) protocols that allow any networked device to communicate and collaborate mutually as peers. JXTA protocols are programming language independent and multiple implementations known as bindings exists for different environments, defining a set of XML messages to coordinate aspects of P2P networking and it is not an application model nor defines the application that can be developed, the technology only defines the protocols and services to give the developer the basic instruments to create a P2P network application. JXTA is developed by Sun Microsystems with an open source Apache license.

The design model employed by the JXTA group is a very flexible and scalable, defining the interfaces for each protocol and service, providing a default implementation, allowing the user to implement its own protocol according to their needs using the provided interfaces.

JXTA objectives to provide a platform to develop P2P systems can be summarized as following:

- The peers must be able to discover other peers.
- Organize into peer groups.
- Publish and discover network resources.
- Establish communication between peers.
- Supervise other peers activities.

A.1 JXTA Architecture

JXTA architecture is divided in layers, in the following we are going to explain each layer and its functionalities.

A.1.1 JXTA Core

The JXTA Core contains the essential elements for all peers to communicate. In this layer it's encapsulated the code that allows the applications to discover peers, create groups, communication protocols and security. All other JXTA features are build in top of the core layer. The elements included in this layer are:

- Peers.
- Peer groups.
- Network Transport.
- Advertisements.
- Peer Identification.
- Protocols: discovery, communication, messages.
- Security and authentication primitives.

A.1.2 Service Layer

The elements of the service layer provides an abstraction of any of the core elements, generally a specific task of a core element, specially a protocol functionality. The service elements provide facilities that are not necessary for the development of the application but are common to several applications. Some services examples can be the resource discovery, authentication, among others.

The services layer includes additional functionality that is continuously being built by the JXTA community (open-source developers working with JXTA project) in addition to services built by the Project JXTA team. Services built on top of the JXTA platform provide specific capabilities that can required by several common P2P applications and can be used to form a complete P2P solution.

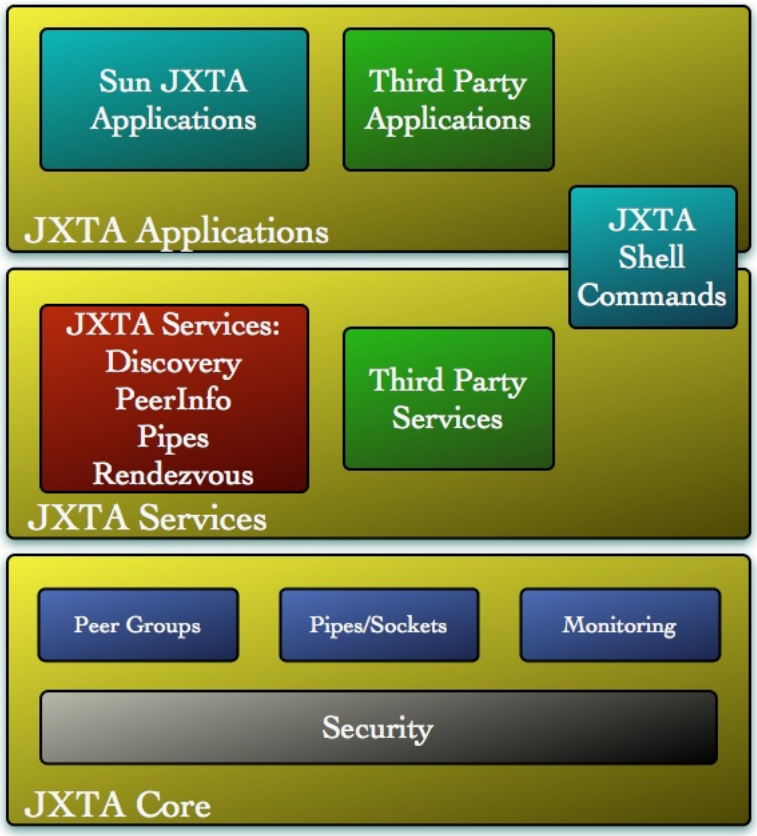

Figure A.1. JXTA Architecture

A.1.3 Application layer

This layer includes the implementation of the common P2P applications like instant messaging. The applications are build in top of services and is difficult to differentiate a service from an application, but the most easy example is one that includes a user interface. The JXTA shell is a good example where we do not have a very specific limit between service and application layer. The JXTA Shell is an application that accepts basic commands from a user interface, this commands are implemented by several simple services, so JXTA is located between the service layer and the application layer in the JXTA architecture.

A.2 JXTA Components

JXTA applications are conformed by several peers that are organized in groups (peer groups) to understand how JXTA works and understand the protocols and services offered by the technology it's necessary to understand the following components and their role.

A.2.1 Peers

Peers is the basic unit of the network, any entity that has access to a networking device that implements the JXTA protocols can be considered as a Peer, some examples, computers, PDAs, sensors, mobile phones. Thanks to the abstraction that offers JXTA in one physical device like a computer we can host several peers, because a peer is not bound to a physical direction or a transport direction, it has its own naming system (each peer has a unique ID) that is going to be explained later. Each peer is independent and asynchronous, and it publishes the available network addresses that are connected to the physical network.

Due to the capacities of a device that hosts a peer and the necessities to establish a direct connection between two peers there are three main types of peers:

- Minimal Edge Peers: this kind of peers only have implemented the core JXTA services and rely on other peers that act as a gateway to access to other services and participate in some application. This minimal edge peers are usually sensors, automatic devices with reduced capabilities.
- Full Edge Peers: the full edge peers implement all the JXTA protocols and services needed to participate in the network, generally are represented by PCs, mobile phones and servers.
- Super-Peers: this peers support and offer resources for the operation of the JXTA network, we have three type of super peers:
- Rendezvous Peer: this peer has the task to maintain indexes of all advertisements and message routing and broadcasting.
- Relay Peer: this peer is in charged of managing connections with peers that doesn't have direct access to the network, for instance due to NAT or firewall.
- Proxy Peer: provides all the services and protocols to minimal edge peers.

A.2.2 Peer Groups

Peer Groups define a group of peers the way they are organized, in order for peers to communicate they must pertain to the same group this creates scopes in very extended networks like Internet. As well as peers Peer Groups are identified with a unique ID allowing a peer to be member of several groups.

By default when a host starts the JXTA platform it has a default bootstrap sequence, at the beginning the peer first instantiates and joins the World-PeerGroup this peer group is only to load the basic services and define the capabilities of the peer, it is important to mention that inside this peer group the peer doesn't have P2P capabilities yet. After joining the WorldPeerGroup the peer instantiates the NetPeerGroup, this is the default peer group for all peers inside this group the peer has the basic services to locate peers, send messages, create peer groups and others.

The idea to create peer groups and divide the peers in groups is, first, to have different scope environments, for instance if I have a file sharing application and an instant messaging application, the idea is that peers that are using the file sharing application don't have to deal with instant messaging application messages or services. Second, a secure environment to restrict access to a group with some security mechanism like username/password or digital certificates and have control over the peers and have different role peers.

A.2.3 Messages

Represent the basic unit to exchange data between peers, messages can be sent through the Endpoint Service or the Pipe Service. Messages are constituted of XML documents, this allows peers to read the information they need by looking only at the interested tags/values.

A.2.4 Pipes

Pipes are the mechanism used by JXTA to exchange messages between peers. Pipes establish a logical connection between two peers and allows to transmit any type of data. The main characteristics is that Pipes are asynchronous, unidirectional and not-reliable. To use a Pipe a peer must bind the endpoint pipe to the peer endpoint, these are known as input endpoint pipe and output endpoint pipe, the peer endpoint corresponds to one of the network transport

layers available on a peer like a TCP/IP network interface used to send the messages.

A.2.5 Services

Peers cooperate and communicate in a group using services, JXTA defines two types of services:

- Peer Services: This services depend on the peer and are implemented in the peer that published the service, if the peer is not available for some reason the service is not available. If there are several peers implementing the same service, each of them publish the service as an independent service.
- Peer Group Services: instances of peer group services are distributed among the peers of the group. It's only necessary to have at least one peer available to access to the service.

JXTA defines a set of basic services, the first two forms part of the core services, for peers to communicate in the same peer group:

- Endpoint Service: allows peers to send and receive messages
- Resolver Service: is used to send generic queries to other peers.
- Discovery Service: allows peers to search for group resources like pipes and other peers.
- Pipe Service: to create and manage pipe connection between peers.
- Membership Service: to establish identities and trust within a group.

A.2.6 Advertisements

Advertisements are XML documents that hold metadata that represent a JXTA resource, entity or service. JXTA offers advertisement for peers, pipes, services, groups, modules and others. Advertisements are used to know about the presence of a resource, they are published locally and remotely using the Discovery service, each advertisement has a validity period when the advertisement expires it is automatically deleted and this indicates that for that peer the resource is not available.

A.2.7 Modules

Modules provide a generic abstraction to allow a peer to instantiate a function or service. When a peer joins a peer group they may find new behaviors that they may want to instantiate. For example, when joining a peer group, a peer may be required to provide a new search service that is only used in this peer group. In order to join this group, the peer must instantiate this new search service. The module framework enables the representation and advertisement of platform-independent behaviors, and allows peers to describe and instantiate any type of implementation of a behavior. For example, a peer has the ability to instantiate either a Java or a C implementation of the specified behavior.

The ability to describe and publish platform-independent behavior is essential to support the development of new peer group services which are provisioned by a heterogeneous cadre of peers. The module advertisement enables JXTA peers to describe a behavior in a platform-independent manner. In fact, JXTA uses module advertisements to self-describe it's own services.

A.2.8 IDs

Peers, peer groups, pipes and other JXTA resources need to be uniquely identifiable. A JXTA ID uniquely identifies a resource and serves as a canonical way of referring to that resource. Currently, there are six types of JXTA entities which have JXTA ID types defined: peers, peer groups, pipes, content, module classes, and module specifications. URNs are used to express JXTA IDs. URNs1 are a form of URI that "... are intended to serve as persistent, location-independent, resource identifiers". Like other forms of URI, JXTA IDs are presented as text.

An example JXTA peer ID is:

`urn:jxta:uuid-59616261646162614A78746150325033F3BC76FF13C2414CBC0AB663666DA53903`

Every JXTA ID has an ID Format. The format describes how the ID was generated and how it may be manipulated by programs. Every ID indicates it's format immediately after the urn:jxta: prefix. There are two common JXTA ID Formats, uuid and jxta, though others exist. The jxta format is used for special common identifiers such as the IDs of the World Peer Group and the Network Peer Group. The uuid format is used for most other IDs. The uuid format provides randomly generated unique IDs and is

based upon DCE GUID/UUID. The portion of a JXTA ID which follows the ID Format is specific to each ID Format and is often opaque.

A.3 JXTA Protocols

In this section are explained the protocols providing background in P2P networking.

All the protocols definedby the JXTA Protocols Specification are implemented as services called core services.The core services include the following:

- Discovery
- Pipe
- Endpoint
- Rendezvous
- Peer Info
- Resolver

An instance of a service is associated with a specific peer group. Only peers that are members of the same peer group are capable of communicating with each other via their services. By default, all peers belong to a common peer group, called Net Peer Group, thereby allowing all peers and their advertisements to be discovered.

Services provide developers with a level of abstraction, insulating them somewhat from the raw message objects used to send information between peers.

A.3.1 Discovery Protocol

The Peer Discovery Protocol consists of only two messages that define the following:

- A request format to use to discover advertisements.
- A response format for responding to a discovery request.

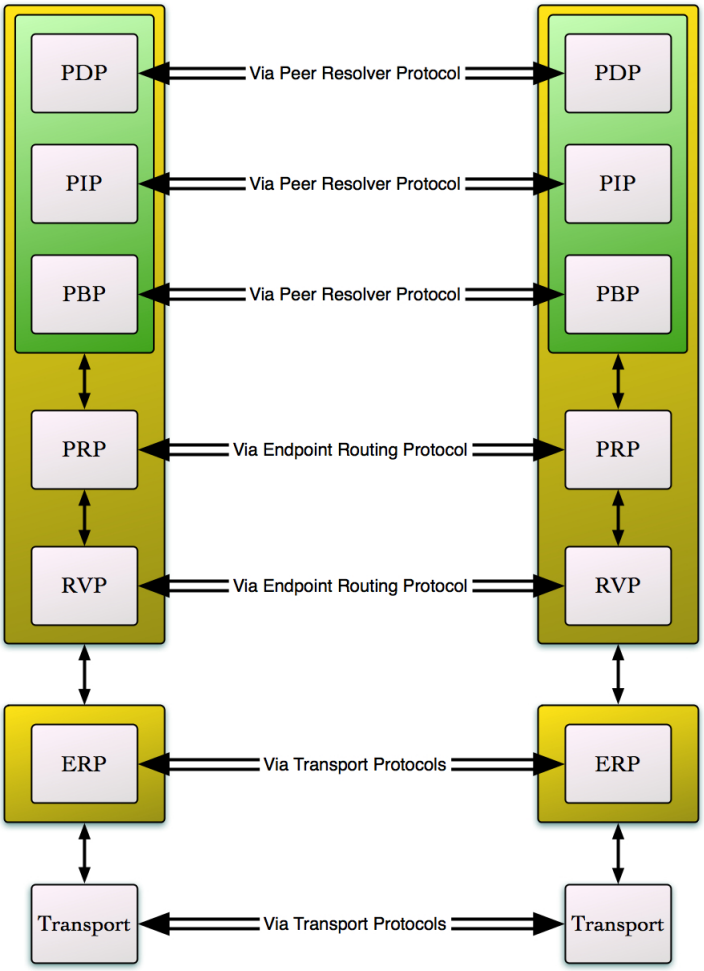

Figure A.2. JXTA Protocol Stack

These two message formats, the Discovery Query Message and the Discovery Response Message, define all the elements required to perform a discovery transaction between two peers. Although the messages define a request and a response to that request, it is important to note that a peer might not expect a Discovery Response Message in response to a given Discovery Query Message. A response to a request might not be received for a variety of reasons, for example, the request didn't generate any results, or the request was ignored by an overloaded peer.

The Discovery service provides a mechanism for the following:

- Retrieving remote advertisements.
- Retrieving local advertisements.
- Publishing advertisements locally.
- Publishing advertisements remotely.
- Flushing local advertisements.

A.3.2 Peer Resolver Protocol

The Peer Resolver Protocol (PRP) defines a protocol for sending a generic query to a named handler located on another peer and processing a generic response to a query. Other services in JXTA, such as the Discovery service, build on the capabilities of the Resolver service and the PRP to provide higher-level services.

The Resolver service is responsible for wrapping a query string in a more generic message format and sending it to a specific handler on a remote peer. In the case of the Discovery service, the query string is the Discovery Query Message and the handler is the remote peer's Discovery service. On the remote peer, a Resolver service instance is responsible for passing an incoming message to the appropriate handler and sending any response generated by the handler.

In the general case, the Resolver service needs only two types of messages:

- Resolver Query Message: A message format for sending queries.
- Resolver Response Message: A message format for sending responses to queries.

These two message formats define generic messages to send queries and responses between peers. At each end, a handler registered with a peer group's Resolver service instance processes query strings and generates response strings.

A.3.3 Rendezvous Protocol

The Rendezvous Protocol (RVP) is used by peers to connect to rendezvous peers with the main objective of propagating messages to other peers on their behalf. The Rendezvous service implementation of the RVP has a dual role, providing a unified API for propagating messages, independent of whether a

peer is configured to act as a rendezvous peer.

Before a peer can use a rendezvous peer to propagate messages, it must connect to the rendezvous peer and obtain a lease. A lease specifies the amount of time that the peer requesting a connection to the rendezvous peer is allowed to use the rendezvous peer before it must renew the connection lease. To handle the interactions required to provide this functionality, the RVP defines three message formats:

- Lease Request Message: A message format used by a peer to request connection lease to the rendezvous peer.
- Lease Granted Message: A message format used by the rendezvous peer to approve a peer's Lease Request Message and provide the length of the lease.
- Lease Cancel Message: A message format used by a peer to disconnect from the rendezvous peer.

A.3.4 Peer Information Protocol

After a remote peer has been discovered using the Discovery service and the Peer Discovery Protocol, a peer might want to monitor the remote peer's status to make additional decisions about how to use the remote peer most effectively or to make the use of its services by other peers more efficient. Monitoring is an essential part of a P2P network, providing information that peers can use to leverage the resources of the P2P network in the most efficient manner. For example, in a file-sharing P2P application, a peer could use status information describing the current network traffic on a remote peer to decide whether to use the remote peer as a source of services. If the remote peer is under an extreme load, it's in the interests of both the client peer and the P2P network in general to shift usage away from that remote peer.

The Peer Information Protocol (PIP) is an optional JXTA protocol that allows a peer to monitor a remote peer and obtain information on the remote peer's current status. PIP uses two types of messages a Peer Info Query and Peer Info Response.

A.3.5 Pipe Binding Protocol

This chapter explains the Pipe Binding Protocol that JXTA peers use to bind a pipe to an endpoint.The PBP defines a set of messages that a peer can

use to query remote peers to find an appropriate endpoint for a given Pipe Advertisement and respond to binding queries from other peers. After a pipe has been bound to an endpoint,a peer can use it to send or receive messages.

The endpoint is the bottom-most element in the network transport abstraction defined by JXTA. Endpoints are encapsulations of the native network interfaces provided by a peer. These network interfaces typically provide access to low-level transport protocols such as TCP or UDP, although some can provide access to higher-level transport protocols such as HTTP. Endpoints are responsible for producing, sending, receiving, and consuming messages sent across the network. Other services in JXTA build on endpoints either directly or indirectly to provide network connectivity. The Resolver service, for example, builds directly on endpoints, whereas the Discovery service builds on endpoints indirectly via the Resolver service.

In addition to the Resolver service, JXTA offers another mechanism by which services can access a network transport without interacting directly with the endpoint abstraction:pipes.Pipes are an abstraction in JXTA that describe a connection between a sending endpoint and one or more receiving endpoints. A pipe is a convenience method layered on top of the endpoint abstraction. Although pipes might appear to provide access to a network transport, implementations of the endpoint abstraction are responsible for the actual task of sending and receiving data over the network. To provide an abstraction that can encompass the simplest networking technology, JXTA specifies pipes as unidirectional, meaning that data travels in only one direction. Pipes are also asynchronous, meaning that data can be sent or received at any time, a feature that allows peers to act independently of other peers without any sort of state synchronization.

A.3.6 Endpoint Routing Protocol

JXTA offers two simpleways to send and receive messages:the Resolver service and the Pipe service. These services are simply convenient wrappers for sending and receiving messages using a peer's local endpoints. An endpoint is an interface to a set of network transports that allows data to be sent across the network. In JXTA, network transports are assumed to be unreliable, even though actual endpoint protocol implementations might use reliable transports such as TCP/IP.

Unlike other areas of the JXTA platform, endpoint functionality doesn't have

a protocol definition. Details on how data is to be formatted for transport across the network is the responsibility of a particular endpoint protocol implementation. The only functionality exposed to the developer is provided by the Endpoint service, which aggregates the registered endpoint protocol implementations for use by a developer. Although a developer could use the Endpoint service implementation to obtain a particular endpoint protocol implementation and use it directly, this is not desirable in most cases. Using a particular endpoint protocol implementation directly makes a solution less flexible by making the solution dependent on a particular network transport.

The Endpoint service provides an access point to all the endpoint protocol implementations installed on a peer, allowing a programmer to send a message using these endpoint protocol implementations. Unlike the other core services in JXTA, the Endpoint service is independent of a peer group. All peer groups share the same Endpoint service, considering that the Endpoint service provides the communication layer closest to the network transport layer. By default, a peer group in the reference implementation inherits the Endpoint service provided by its parent group. However, developers can provide a custom Endpoint service implementation for a peer group that they create by loading a custom Endpoint service.

Appendix B

Server Code

```
package server;

import java.net.*;
import java.io.*;
import java.util.*;

public class ServerControl implements Runnable  {

// Variables for each thread
Socket linkto; // the socket
PrintWriter outputBuffered; // the output and input streams
BufferedReader inputBuffered;
int id; // i.d. of the connection
String from_name; // name of host connecting
InputStream in;
BufferedReader inBuf;
PrintWriter outBuf;
OutputStream out;
InputStream inData;
OutputStream outData;
InetAddress ip;
// Class Variables

static Vector connectiontable; // current connections
static int nextid = 1; // Increasing i.d.

////////////////////////////// Main method ////////////////////////

public static void main(String [] args) {
// Parent thread - create a server socket and await a connection
ServerSocket ss = null;
Socket s = null;
connectiontable = new Vector();

try {
ss = new ServerSocket(1357);
while ((s=ss.accept())!= null) {

// Connection received - create a thread
ServerControl now;
Thread current = new Thread(now = new ServerControl(s));
```

```
//current.setDaemon(true);
connectiontable.addElement(now); // Save talker into vector ..
current.start(); // start the user's thread
}
} catch (Exception e) {
System.out.println(e);
}

// should add finally block to close down
}
///////////////// Constructor for a new thread ////////////////////

ServerControl (Socket from) {
id = nextid++;
linkto = from;
ip = linkto.getInetAddress();
InetAddress source = linkto.getInetAddress();
from_name = source.getHostName();
try {

out = linkto.getOutputStream();

in = linkto.getInputStream();

inBuf = new BufferedReader(new InputStreamReader(in));
outBuf = new PrintWriter(new OutputStreamWriter(out));
}
catch (Exception e) {}
}

public void run () {

String line = null;

byte buf[] = new byte[65536];
//byte buf[] = new byte[65536];
byte rbuf[] = new byte[65536];
byte udpBuf[] = new byte[1472];
byte[] entero = new byte[4];
byte[] largo = new byte[8];
int b = 0;
int download = 0;
int serverPort = 0;
int serverUdpPort = 0;
int clientUdpPort = 0;
int totalData = 65536*32;
int data = 0;
Socket s = null;
ServerSocket socket;
DatagramSocket udpSocket;

//Fill buffer with random data
ServerControl.fillBuffer(buf);

try {
socket = new ServerSocket(serverPort =
ServerControl.generateRandom());
outBuf.println(serverPort);
outBuf.flush();

if ((s = socket.accept())!= null) {
```

```
outBuf.println("Connected to: " + s.getLocalPort() + " from " +
s.getInetAddress().getHostAddress() );
outBuf.flush();
}

//Obtain streams to send and receive bulk data
inData = s.getInputStream();
outData = s.getOutputStream();

} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}


while (true) {
boolean done=false;

// read a line from the user

if (line == null) {
try {

System.out.println("Instruction " + (b = Integer.parseInt(
inBuf.readLine())));

} catch (Exception e) {
System.out.println(e);
done = true; // force exit if there's a problem
}
}

if ( b == 4 ) {
long init_total = 0;
int total_speed = 0;
System.out.println("UDP Port: " + (serverUdpPort =
ServerControl.generateRandom()));
ServerControl.fillBuffer(udpBuf);
try {

int udpTotal = 0;
outBuf.println(serverUdpPort);
outBuf.flush();

//Read Client UDP Port
clientUdpPort = Integer.parseInt(inBuf.readLine());
System.out.println("Client Udp Port: " + clientUdpPort);
udpSocket = new DatagramSocket(serverUdpPort);
//udpSocket.connect(ip, clientUdpPort);
udpSocket.setSoTimeout(5000);

//Receive Bulk Data

udpTotal = Integer.parseInt(inBuf.readLine());
System.out.println("udpTotal: " + udpTotal);
DatagramPacket p = new DatagramPacket(udpBuf, udpBuf.length);
int totalUdp = 0;
outBuf.println("Ready");
outBuf.flush();
System.out.println(udpSocket.getRemoteSocketAddress());
int actualTotal = 0;
init_total = System.nanoTime();
```

```
try {
udpSocket.receive(p);
clientUdpPort = p.getPort();
if (p.getAddress().equals(ip)) udpSocket.connect(ip,
clientUdpPort);

totalUdp = totalUdp + 1472;
actualTotal = actualTotal + 1472;
while (actualTotal < udpTotal){
udpSocket.receive(p);
//totalUdp = totalUdp + 1472;
actualTotal = actualTotal + 1472;
}
} catch (SocketTimeoutException e) {
totalUdp = udpTotal; //To exit loop
//break;//When losing a segment stop the test
}
long end_total = System.nanoTime();
outBuf.println("Finish");
outBuf.flush();
outBuf.println(actualTotal);
outBuf.flush();
System.out.println(actualTotal);
long delta_total = end_total - init_total;
//System.out.println(delta_total);
System.out.println("Download Speed: " + (total_speed = (int)
(8*actualTotal/(1024*delta_total/Math.pow(10, 9)))));
System.out.println(Integer.parseInt(inBuf.readLine()));



outBuf.println(2194304);
outBuf.flush();

DatagramPacket packet = new DatagramPacket(udpBuf,
udpBuf.length, ip, clientUdpPort);
//udpSocket.receive(packet);


if (inBuf.readLine().startsWith("Ready")){
System.out.println("Entre");
try {
Thread.sleep(2000);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

udpTotal = 0;
init_total = System.nanoTime();

while( udpTotal < 2194304) {

udpTotal = udpTotal + 1472;
udpSocket.send(packet);
ServerControl.fillBuffer(udpBuf);
packet.setData(udpBuf);
try {
Thread.sleep(0,1);
} catch (InterruptedException e) {
```

```
// TODO Auto-generated catch block
e.printStackTrace();
}

}}
if (inBuf.readLine().startsWith("Finish")) {
end_total = System.nanoTime();
int remoteTotal = Integer.parseInt(inBuf.readLine());
System.out.println("Speed: " + (total_speed = (int)
(8*remoteTotal/(1024*(end_total-init_total)/Math.pow(10, 9)))));
outBuf.println(total_speed);
outBuf.flush();
}



udpSocket.disconnect();
udpSocket.close();


} catch (SocketException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (NumberFormatException e) {
System.out.println("Error");
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

done = true;

}

//Quit Finish Thread
if ( b == 3 ) {
try {
outData.close(); // closes needed to terminate connection
inData.close(); // otherwise user's window goes mute
s.close();
} catch (IOException e) {
}
//done = true;
}


//Round trip
if ( b == 0 ) {

try {

System.out.println(inBuf.readLine());
outBuf.println(System.currentTimeMillis());
outBuf.flush();

} catch (Exception e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}
}

//Download Test
```

```
if ( b == 1 ) {

try {
//Send Buffer length

outBuf.println(totalData);
outBuf.flush();

try {
Thread.sleep(10);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

//Send Bulk Data
while (data < totalData) {
outData.write(buf);
outData.flush();
data += buf.length;
ServerControl.fillBuffer(buf);
}
download = Integer.parseInt(inBuf.readLine());
System.out.println("Download Speed: " + download);

} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}

//Upoload Test
if( b == 2 ) {
int bytes;
long init_total;
boolean continuar = true;
int total = 0;
long end_total = 0;
long delta_total = 0;
int total_speed = 0;
int ch = 0;
int i = 0;
int bufLength = 0;
byte[] b1 = new byte[8];

try {
System.out.println(bufLength = Integer.parseInt(inBuf.readLine()));

init_total = System.nanoTime();
while(continuar) {

ch = inData.read(rbuf, 0, 65536);
total+=ch;
if (ch == -1 || total == bufLength) {
end_total = System.nanoTime();
System.out.println("no continue");
continuar = false;
}
}

delta_total = end_total - init_total;
System.out.println(total);
```

```
System.out.println("Speed: " + (total_speed = (int) (
8*total/(1024*delta_total/Math.pow(10, 9)))));
outBuf.println(total_speed);
outBuf.flush();


} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}

// clear out the user if they're done

if (done) {
connectiontable.removeElement(this);
break;
}
line = null;
}

//System.out.println("Finishing Thread");
}

public static void fillBuffer(byte buf[]) {

Random random = new Random();
random.nextBytes(buf);
/*for (int i = 0; i < buf.length; i++) {
System.out.println(buf[i]);
}*/
}

public static int generateRandom() {
  int port = 0;
  Random random = new Random();
  port = 49152 + random.nextInt(65535 - 49152);
  return port;
  }
}
```

Appendix C

Client Code

```
package client;

import java.net.*;
import java.awt.Toolkit;
import java.io.*;
import java.util.*;

import javax.swing.JLabel;
import javax.swing.JProgressBar;
import javax.swing.JTextArea;
import javax.swing.ProgressMonitor;

import org.jdesktop.swingworker.SwingWorker;

import server.ServerControl;

public class ClientControl extends SwingWorker {

private Socket controlSocket;
private Socket socket;
private byte[] dir = {(byte) 130, (byte) 192,(byte) 47,(byte) 70};
private OutputStream out;

private InputStream in;
private long init;
private long end;
private int total;
byte receiveBuf[] = new byte[65536];
byte sendBuf[] = new byte[65536];
private JProgressBar bar;
private JTextArea taskOutput;
private BufferedReader inBuf;
private PrintWriter outBuf;
private InputStream inData;
private OutputStream outData;
private DatagramSocket udpSocket;
private int serverUdpPort = 0;
private int udpPort = 0;
private int progress = 0;
private SpeedData speedData;
private JLabel downloadLabel;
```

```
private JLabel uploadLabel;

public ClientControl (int i ) {
this.bar = bar;
this.taskOutput = taskOutput;
ServerControl.fillBuffer(sendBuf);
}

public ClientControl (int i, JTextArea taskOutput,
JProgressBar bar, JLabel jLabel3, JLabel jLabel6) {
this.bar = bar;
this.taskOutput = taskOutput;
this.downloadLabel = jLabel3;
this.uploadLabel = jLabel6;
this.speedData = new SpeedData();
ServerControl.fillBuffer(sendBuf);
}

public class SpeedData {

private String ip;
private int tcpDown, tcpUp, udpDown, udpUp, serverControl,
clientControl, tcpServer, tcpClient, udpServer, udpClient;
long rtt;
long conTime;
public String getIp() {
return ip;
}
public void setIp(String ip) {
this.ip = ip;
}
public int getTcpDown() {
return tcpDown;
}
public void setTcpDown(int tcpDown) {
this.tcpDown = tcpDown;
}
public int getTcpUp() {
return tcpUp;
}
public void setTcpUp(int tcpUp) {
this.tcpUp = tcpUp;
}
public int getUdpDown() {
return udpDown;
}
public void setUdpDown(int udpDown) {
this.udpDown = udpDown;
}
public int getUdpUp() {
return udpUp;
}
public void setUdpUp(int udpUp) {
this.udpUp = udpUp;
}
public long getRtt() {
return rtt;
}
public void setRtt(long rtt2) {
this.rtt = rtt2;
}
public long getConTime() {
```

```
return conTime;
}
public void setConTime(long l) {
this.conTime = l;
}
public int getServerControl() {
return serverControl;
}
public void setServerControl(int serverControl) {
this.serverControl = serverControl;
}
public int getClientControl() {
return clientControl;
}
public void setClientControl(int clientControl) {
this.clientControl = clientControl;
}
public int getTcpServer() {
return tcpServer;
}
public void setTcpServer(int tcpServer) {
this.tcpServer = tcpServer;
}
public int getTcpClient() {
return tcpClient;
}
public void setTcpClient(int tcpClient) {
this.tcpClient = tcpClient;
}
public int getUdpServer() {
return udpServer;
}
public void setUdpServer(int udpServer) {
this.udpServer = udpServer;
}
public int getUdpClient() {
return udpClient;
}
public void setUdpClient(int udpClient) {
this.udpClient = udpClient;
}
}


public static void main(String args[]) {

Timer timer = new Timer();
//timer.scheduleAtFixedRate(new ClientControl(2), 0, 60*1000);
}



@Override
protected Object doInBackground() throws Exception {
// TODO Auto-generated method stub

bar.setIndeterminate(true);

        int serverPort = 0;
        String line;
        long beginPing = 0;
        long endPing = 0;
```

```
        long rtt = 0;
      total = 0;
      int ch = 0;
      long end_total = 0;
      long delta_total = 0;
      long init_total = 0;
      int total_speed = 0;
      int instruction = 0;
      int total_speed1 = 0;
      long beginrt = 0;
      int bufLength = 0;
      int udpTotal = 0;
      int totalUdp = 0;
byte[] udpBuf  = new byte[1472];
int data = 0;
int totalData = 65536*64;


try {
init = System.currentTimeMillis();
controlSocket = new Socket( InetAddress.getByName(
"neubotpc.polito.it"), 1357 );
//controlSocket = new Socket( InetAddress.getByAddress(dir), 1357 );
speedData.setServerControl(1357);
speedData.setIp(controlSocket.getInetAddress().getHostAddress());

if (controlSocket != null) {
end = System.currentTimeMillis();
speedData.setConTime((end - init));
speedData.setClientControl(controlSocket.getLocalPort());
System.out.println("Connected to control socket in: " + (end-init));
} else System.out.println("Connection failed");



out = controlSocket.getOutputStream();
in = controlSocket.getInputStream();
outBuf = new PrintWriter(new OutputStreamWriter(out));
inBuf = new BufferedReader(new InputStreamReader(in));
System.out.println("Leyendo");
line = inBuf.readLine();
serverPort = Integer.parseInt(line);
System.out.println("Puerto " + serverPort);
speedData.setTcpServer(serverPort);

init = System.currentTimeMillis();


/*try {
Thread.sleep(500);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}*/

socket = new Socket( InetAddress.getByName("neubotpc.polito.it"),
serverPort );
//socket = new Socket( InetAddress.getByAddress(dir), serverPort );

if (socket != null) {
end = System.currentTimeMillis();
speedData.setTcpClient(socket.getLocalPort());
```

```
System.out.println("Connected to socket in: " + (end-init));
inData = socket.getInputStream();
outData = socket.getOutputStream();
//socket.setSoTimeout(1000);

} else System.out.println("Connection failed");

} catch (UnknownHostException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

try {
if ( inBuf.readLine().startsWith("Connected") ){

while (instruction < 5) {

outBuf.println(instruction);
System.out.println("Instruction: " + instruction);
outBuf.flush();
System.out.println("Request Sent");

switch (instruction) {

case 0:

beginPing = System.currentTimeMillis();
outBuf.println(beginPing);
outBuf.flush();
System.out.println(beginrt = Long.parseLong(inBuf.readLine()));
//Thread.sleep(100);
endPing = System.currentTimeMillis();
System.out.println("latecny: " + (endPing-beginrt));
System.out.println("rtt: " + (rtt = (endPing-beginPing)));
taskOutput.append("RTT: " + rtt + " msec \n");
instruction ++;
speedData.setRtt(rtt);
//Thread.sleep(1000);
break;

case 1:

bar.setIndeterminate(false);
System.out.println(bufLength = Integer.parseInt(
inBuf.readLine()));

boolean continuar = true;

init_total = System.nanoTime();

while(continuar) {

ch = inData.read(receiveBuf, 0, 65536);
total+=ch;
progress = progress + total;
progress = progress/(bufLength/25);
System.out.println (progress+ " " + total );
setProgress(Math.min(progress,100));
if (ch == -1 || total == bufLength) {
```

```
end_total = System.nanoTime();
System.out.println("no continue");
continuar = false;
}
//progress = total/4194304;
//setProgress(Math.min(progress, 100));
}
instruction ++;
//monitor.setNote("Upload Test");

System.out.println("Termine");
delta_total = end_total - init_total;
System.out.println(total);
System.out.println("Download Speed: " + (total_speed = (int)
(8*total/(1024*delta_total/Math.pow(10, 9)))));
outBuf.println(total_speed);
outBuf.flush();
taskOutput.append("Download Speed: " + total_speed + " kbps \n");
speedData.setTcpDown(total_speed);
break;

case 2:

try {
//Send Buffer length
outBuf.println(totalData);
outBuf.flush();

//Send Bulk Data
System.out.println(progress);
//Send Bulk Data
while (data < totalData) {
progress = progress + data;
progress = 25 + progress/((totalData/25));
System.out.println(progress + " " + data);
setProgress(Math.min(progress,100));
outData.write(sendBuf);
outData.flush();
data += sendBuf.length;
ServerControl.fillBuffer(sendBuf);
}

total_speed1 = Integer.parseInt(inBuf.readLine());
System.out.println("Upload Speed: " + total_speed1);
taskOutput.append("Upload Speed: " + total_speed1 + " kbps \n");
speedData.setTcpUp(total_speed1);

} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

bar.setIndeterminate(true);
instruction ++;

break;

case 3:

inData.close();
outData.close();
socket.close();
```

```
System.out.println("Connection Closed");

instruction ++;
break;

case 4:

//setProgress(50);
serverUdpPort = Integer.parseInt(inBuf.readLine());
udpPort = ServerControl.generateRandom();
outBuf.println(udpPort);
outBuf.flush();
udpSocket = new DatagramSocket(udpPort);
udpSocket.setSoTimeout(10000);
udpSocket.connect( InetAddress.getByName("neubotpc.polito.it"),
serverUdpPort);
//Send Bulk Data (UpLoad)
speedData.setUdpClient(udpPort);
ServerControl.fillBuffer(udpBuf);
DatagramPacket packet = new DatagramPacket(udpBuf, udpBuf.length,
 InetAddress.getByAddress(dir), serverUdpPort );
outBuf.println(2194304);
outBuf.flush();

if (inBuf.readLine().startsWith("Ready")){
System.out.println("Entre");
try {
Thread.sleep(2000);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
bar.setIndeterminate(false);
udpTotal = 0;
udpSocket.connect(udpSocket.getRemoteSocketAddress());
init_total = System.nanoTime();
while( udpTotal < 2194304 ) {
progress = progress + udpTotal;
progress = 50 + progress/((2194304/25));
setProgress(progress);
udpTotal = udpTotal + 1472;
udpSocket.send(packet);
ServerControl.fillBuffer(udpBuf);
packet.setData(udpBuf);
//OJO QUITAAAAAR
//Thread.sleep(1);
}
System.out.println(udpSocket.getRemoteSocketAddress());
speedData.setUdpServer(udpSocket.getPort());
}
bar.setIndeterminate(true);

if (inBuf.readLine().startsWith("Finish")) {
end_total = System.nanoTime();
int remoteTotal = Integer.parseInt(inBuf.readLine());
System.out.println("Upload Speed: " + (total_speed = (int)
(8*remoteTotal/(1024*(end_total-init_total)/Math.pow(10, 9)))));
taskOutput.append("UDP Upload Speed: " + total_speed + " kbps \n");
speedData.setUdpUp(total_speed);
outBuf.println(total_speed);
outBuf.flush();
}
```

```
//Receive Bulk Data

udpTotal = Integer.parseInt(inBuf.readLine());
System.out.println("udpTotal: " + udpTotal);
DatagramPacket p = new DatagramPacket(udpBuf, udpBuf.length);
totalUdp = 0;
outBuf.println("Ready");
outBuf.flush();
System.out.println(udpSocket.getRemoteSocketAddress());
int actualTotal = 0;
bar.setIndeterminate(false);
init_total = System.nanoTime();


try {
udpSocket.receive(p);
totalUdp = totalUdp + 1472;
actualTotal = actualTotal + 1472;
udpSocket.setSoTimeout(100);

while (actualTotal < udpTotal){
progress = progress + actualTotal;
progress = 75 + progress/((2194304/25));
setProgress(progress);
udpSocket.receive(p);
//totalUdp = totalUdp + 1472;
actualTotal = actualTotal + 1472;
System.out.println(actualTotal);
}
end_total = System.nanoTime();
} catch (SocketTimeoutException e) {
totalUdp = udpTotal; //To exit loop
end_total = System.nanoTime();

//break;//When losing a segment stop the test
}
//end_total = System.nanoTime();
outBuf.println("Finish");
outBuf.flush();
outBuf.println(actualTotal);
outBuf.flush();
System.out.println(actualTotal);
delta_total = end_total - init_total;
//System.out.println(delta_total);
System.out.println("Download Speed: " + (total_speed = (int)
(8*actualTotal/(1024*delta_total/Math.pow(10, 9)))));
System.out.println(total_speed1 = Integer.parseInt(
inBuf.readLine()));
speedData.setUdpDown(total_speed1);
taskOutput.append(
"UDP Download Speed: " + total_speed1 + " kbps \n");


udpSocket.disconnect();
udpSocket.close();
instruction ++;
setProgress(100);
```

```
break;

default: System.out.println(
"Instruction not valid. Refused by the Server"); break;

}
}

downloadLabel.setText(Integer.toString(speedData.getTcpDown()));
uploadLabel.setText(Integer.toString(speedData.getTcpUp()));

System.out.println(speedData.ip);
System.out.println(speedData.conTime);
System.out.println(speedData.serverControl);
System.out.println(speedData.tcpClient);
System.out.println(speedData.tcpDown);
System.out.println(speedData.tcpServer);
System.out.println(speedData.tcpUp);
System.out.println(speedData.udpClient);
System.out.println(speedData.udpDown);
System.out.println(speedData.udpServer);
System.out.println(speedData.udpUp);


DisplayTable dis = new DisplayTable();
dis.insertSpeedTest(speedData);

//taskOutput.append("Download Speed: " + total_speed + " kbps\n");
//taskOutput.append("Upload Speed: " + total_speed1 + "kbps\n");
}
} catch (NumberFormatException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (SocketException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (UnknownHostException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
        //return null;
return null;
}

}
```

Appendix D

Network Configurator JXTA

```
package jxta.defaultconfig.config;


import net.jxta.exception.PeerGroupException;
import net.jxta.peergroup.PeerGroup;
import net.jxta.platform.NetworkConfigurator;
import net.jxta.platform.NetworkManager;

import gooee.NeubotGui;

import java.io.File;
import java.io.IOException;
import java.net.URI;
//import java.net.URI;


import javax.security.cert.CertificateException;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;


public class DefaultConfig {

public NetworkManager netManager = null;

private NetworkConfigurator netConfig = null;

public static final String Local_Network_Manager_Name =
"NEUBOT Network Manager";

public static final String Local_Peer_Name = System.getProperty
("user.name");

private PeerGroup netPeerGroup = null;

private JTextArea stdOut;

private JScrollPane stdOutScroll;


public DefaultConfig(JTextArea stdOut, JScrollPane stdOutScroll) {
```

```
// Creating the Network Manager
try {
this.stdOut = stdOut;
this.stdOutScroll = stdOutScroll;
NeubotGui.appendText(stdOutScroll, stdOut,
"Creating the Network Manager \n");
//stdOut.append("Creating the Network Manager \n");
System.out.println("Creating the Network Manager");

netManager = new NetworkManager(
NetworkManager.ConfigMode.EDGE,
Local_Network_Manager_Name,
new File(new File(".config"), "NEUBOT").toURI());

NeubotGui.appendText(stdOutScroll, stdOut,
"Network Manager Created \n");
//stdOut.append("Network Manager Created \n");
System.out.println("Network Manager created");

} catch (IOException ex) {
ex.printStackTrace();
System.exit(-1);
}

// Persisting it to make sure the Peer ID is not re-created each
// time the Network Manager is instantiated
netManager.setConfigPersistent(true);

NeubotGui.appendText(stdOutScroll, stdOut, "PeerID: " +
netManager.getPeerID().toString() + "\n");
System.out.println("PeerID: "
+ netManager.getPeerID().toString());

NeubotGui.appendText(stdOutScroll, stdOut, "Network ID: " +
netManager.getInfrastructureID().toString() + "\n");
System.out.println("Network ID: "
+ netManager.getInfrastructureID().toString());

// Since we won't be setting our own relay or rendezvous seed peers we
// will use the default (public network) relay and rendezvous seeding.
netManager.setUseDefaultSeeds(false);

// Retrieving the Network Configurator
NeubotGui.appendText(stdOutScroll, stdOut,
"Retrieving Network Configurator \n");
//stdOut.append("Retrieving Network Configurator \n");
System.out.println("Retrieving the Network Configurator");

try {
netConfig = netManager.getConfigurator();
} catch (IOException e) {
e.printStackTrace();
}

NeubotGui.appendText(stdOutScroll, stdOut,
"Network Configurator Retrieved \n");
//stdOut.append("Network Configurator Retrieved \n");
System.out.println("Network Configurator retrieved");

// Does a local peer configuration exist?
if (netConfig.exists()) {
```

```
NeubotGui.appendText(stdOutScroll, stdOut,
"Local Configuration Found \n");
//stdOut.append("Local Configuration Found \n");
System.out.println("Local configuration found");

// We load it
File LocalConfig = new File(netConfig.getHome(),
"PlatformConfig");

try {
NeubotGui.appendText(stdOutScroll, stdOut,
"Loading Local Configuration \n");
//stdOut.append("Loading Local Configuration \n");
System.out.println("Loading found configuration");
netConfig.load(LocalConfig.toURI());
System.out.println("Configuration loaded");
System.out.println("Home directory: " +
netConfig.getHome());

} catch (IOException ex) {
ex.printStackTrace();
System.exit(-1);

} catch (CertificateException ex) {
// An issue with the existing peer certificate has been
// encountered
ex.printStackTrace();
System.exit(-1);
}

} else {

NeubotGui.appendText(stdOutScroll, stdOut,
"Local Configuration not found \n");
//stdOut.append("Local Configuration not Found \n");
System.out.println("No local configuration found");

//Default Configuration Parameters
netConfig.setName(Local_Peer_Name);
netConfig.setPrincipal("12345678");
netConfig.setPassword("12345678");
netConfig.setUseMulticast(false);
netConfig.setHttpEnabled(true);
netConfig.addSeedRendezvous(
URI.create("neubotpc.polito.it"));

System.out.println("Home directory: " + netConfig.getHome());
System.out.println("Name: " + netConfig.getName());
System.out.println("Principal: " + netConfig.getPrincipal());
System.out.println("Password : " + netConfig.getPassword());

try {

NeubotGui.appendText(stdOutScroll, stdOut,
"Saving the New Configuration \n");
//stdOut.append("Saving the New Configuration \n");
System.out.println("Saving new configuration");

netConfig.save();

NeubotGui.appendText(stdOutScroll, stdOut,
"New Configuration Saved Successfully \n");
```

```
//stdOut.append("New Configuration Saved Successfully \n");
System.out.println("New configuration saved successfully");

} catch (IOException ex) {
ex.printStackTrace();
System.exit(-1);
}
}
}

public PeerGroup startJxta() {

try {

NeubotGui.appendText(stdOutScroll, stdOut, "Starting JXTA \n");
//stdOut.append("Starting JXTA \n");
System.out.println("Starting JXTA");

netPeerGroup = netManager.startNetwork();

NeubotGui.appendText(stdOutScroll, stdOut, "JXTA Started \n");
//stdOut.append("JXTA Started \n");
System.out.println("JXTA Started");

NeubotGui.appendText(stdOutScroll, stdOut, "Peer Name: " +
netPeerGroup.getPeerName() + "\n");
//stdOut.append("Peer Name: " + netPeerGroup.getPeerName() + "\n");
System.out.println("Peer name      : "
+ netPeerGroup.getPeerName());

NeubotGui.appendText(stdOutScroll, stdOut, "Peer Group Name: " +
netPeerGroup.getPeerGroupName() +"\n");
System.out.println("Peer Group name: "
+ netPeerGroup.getPeerGroupName());

NeubotGui.appendText(stdOutScroll, stdOut, "Peer ID: " +
netPeerGroup.getPeerID().toString() + "\n");
System.out.println("Peer ID  : "
+ netPeerGroup.getPeerID().toString());

NeubotGui.appendText(stdOutScroll, stdOut,
"Waiting For a Rendezvous Connection \n");
//stdOut.append("Waiting For a Rendezvous Connection \n");
System.out.println("Waiting for a Rendezvous Connection");

netManager.waitForRendezvousConnection(10000);

} catch (PeerGroupException ex) {
// Cannot initialize peer group
ex.printStackTrace();
System.exit(-1);
} catch (IOException ex) {
ex.printStackTrace();
System.exit(-1);
}

return netPeerGroup;

}

public void stopJxta() {
```

```
netManager.stopNetwork();
}

}
```

Appendix E

Presence Service: Code extract

```
package jxta.servicetest;

import java.io.IOException;
import java.net.SocketException;
import java.util.Random;
import java.util.Vector;

import javax.swing.JScrollPane;
import javax.swing.JTextArea;

import jxta.defaultconfig.createadvertisement.PipeAdv;
import jxta.defaultconfig.propagatepipe.ServerPipe;
import jxta.services.presence.PresenceMessages;
import jxta.services.presence.messagehandler.MessageHandler;

import net.jxta.document.Advertisement;
import net.jxta.document.MimeMediaType;
import net.jxta.document.XMLDocument;
import net.jxta.endpoint.Message;
import net.jxta.endpoint.StringMessageElement;
import net.jxta.endpoint.TextDocumentMessageElement;
import net.jxta.exception.PeerGroupException;
import net.jxta.id.IDFactory;
//import net.jxta.id.ID;
import net.jxta.peergroup.PeerGroup;
import net.jxta.pipe.InputPipe;
import net.jxta.pipe.OutputPipe;
import net.jxta.pipe.PipeID;
import net.jxta.pipe.PipeService;
import net.jxta.util.JxtaServerPipe;
import net.jxta.util.JxtaBiDiPipe;
//import net.jxta.protocol.ModuleImplAdvertisement;
import net.jxta.protocol.PipeAdvertisement;

public class FirstServiceImpl implements FirstService {

private Advertisement implAdvertisement = null;

private PipeAdvertisement propagatePipeAdv = null;

private boolean isRendezvous = false;
```

```
private InputPipe inPipe = null;

private OutputPipe outPipe = null;

private JxtaBiDiPipe biDiPipe = null;

private MessageHandler mHandler = null;

private String ipDir = null;

private String peerState = "IDDLE";

//private String handlerName = null;

private Vector<FirstServiceListenet> registeredListeners =
new Vector<FirstServiceListenet>();

private PeerGroup neubotGroup = null;

private JTextArea stdOut;

private JScrollPane stdOutScroll;

public MessageHandler getMessageHandler() {
return mHandler;
}

public void setPeerState(String state) {
this.peerState = state;
}

public String getPeerState() {
return this.peerState;
}

public PeerGroup getNeubotGroup () {
return neubotGroup;
}

public JxtaBiDiPipe getBiDiPipe () {
return biDiPipe;
}
//private ResolverService resolverService = null;

//private int queryID = 0;

public void addListener(FirstServiceListenet listener) {
registeredListeners.addElement(listener);
}

public void removeListener(FirstServiceListenet listener) {
registeredListeners.removeElement(listener);
}

public Advertisement getImplAdvertisement() {
return implAdvertisement;
}

/*public Service getInterface() {
// TODO Auto-generated method stub
return this;
```

```
}*/

public void init(PeerGroup neubotGroup, boolean isRendezvous,
String ipDir, JTextArea stdOut, JScrollPane stdOutScroll
) throws PeerGroupException {
// TODO Auto-generated method stub
//implAdvertisement = (ModuleImplAdvertisement)implAdv;

//handlerName = assignedID.toString();
this.ipDir = ipDir;
this.neubotGroup = neubotGroup;
stdOut.append("Presence Service Initialized \n");
System.out.println("Service initialized");
this.isRendezvous = isRendezvous;
this.stdOut = stdOut;
this.stdOutScroll = stdOutScroll;

}

public int startApp(String[] arg0) {
// TODO Auto-generated method stub


PipeService pipeService = null;

pipeService = neubotGroup.getPipeService();

//Creation of a Propagate Pipe Advertisement to create the Input pipe
PipeAdv pipeAdv = new PipeAdv(neubotGroup, PipeService.PropagateType,
"urn:jxta:uuid-F303C7499B09477FABD2A7F0E436C86EAC17BF0EC0EE4F60A6D4F887D5682DAA04");

propagatePipeAdv = pipeAdv.generatePipeAdv();

mHandler = new MessageHandler(this,
(PipeID)pipeAdv.pipeAdv.getID(), null, ipDir, stdOut, stdOutScroll);

System.out.println(isRendezvous);

if (isRendezvous) {

/*If the peer is a Rendezvous it should start as a Listening Peer
 * creating an Input Propagated pipe in order to have
 * a bootstrap peer a Rendezvous peer shouldn't be interested in
 * doing measurements
 */
try {

inPipe = pipeService.createInputPipe(propagatePipeAdv, mHandler);

if (inPipe != null) {
stdOut.append("Input Pipe Created \n");
System.out.println("Input Pipe Created");
}

} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
stdOut.append("Couldn't Create Input Pipe. This is likely a bug \n");
System.out.println("Couldn't Create Input Pipe");
}
}
else {
```

```
/*
 * For the rest of the peers (Generally EDGE peers) they begin sending
 * a request message through the Output Propagated Pipe to locate
 * a peer to do the measures
 */
PipeAdv serverPipeAdvFactory = new PipeAdv(neubotGroup, PipeService.UnicastType,
IDFactory.newPipeID(neubotGroup.getPeerGroupID()).toString());

PipeAdvertisement serverPipeAdv = serverPipeAdvFactory.generatePipeAdv();

try {

outPipe = pipeService.createOutputPipe(propagatePipeAdv, 10000);

if (outPipe != null) {
stdOut.append("Output Pipe Created \n");
System.out.println("Output Pipe Created");
}

/*
 * Now we send a message with a server bidipipe advertisement to obtain
 * a peer interested to do the measurements
 */

Message message = new Message();

message.addMessageElement("NeubotPresence", new
TextDocumentMessageElement("PipeAdv", (XMLDocument)
serverPipeAdv.getDocument(MimeMediaType.XMLUTF8), null));

message.addMessageElement("NeubotPresence",
new StringMessageElement("SenderPeerID",
neubotGroup.getPeerID().toString(), null));

outPipe.send(message);

outPipe.close();

try {
Thread.sleep(1000);
} catch (InterruptedException e2) {
// TODO Auto-generated catch block
e2.printStackTrace();
}
/*
 * Now we create the ServerPipe to wait for incoming conections
 */

ServerPipe serverPipeFactory = new ServerPipe(neubotGroup, serverPipeAdv);

JxtaServerPipe serverPipe = serverPipeFactory.createServerPipe();

try {
serverPipe.setPipeTimeout(10000);
} catch (SocketException e) {
// TODO Auto-generated catch block
e.printStackTrace();
stdOut.append("Error Setting the BiDiPipe server timeout \n");
System.out.println("Timeout Error");
}
```

```
try{

stdOut.append("Waiting for JXTA BiDiPipe Connections \n");
System.out.println("Waiting for JXTA BiDiPipe Connections");

biDiPipe = serverPipe.accept();

biDiPipe.setMessageListener(mHandler);

if (biDiPipe != null) {

peerState = "PAIRED";
stdOut.append("Peer State: " + peerState + " \n");
System.out.println(peerState);

try {

Thread.sleep(1000);

} catch (InterruptedException e1) {

e1.printStackTrace();

}

stdOut.append("BiDiPipe Accepted \n");
System.out.println("Jxta BiDiPipe accepted");

stdOut.append("Sending Protocol Request Message \n");
System.out.println("Sending Request Protocol Message");

Random randomGenerator = new Random();

int code = randomGenerator.nextInt(65535);

mHandler.setSequenceCode(code);
biDiPipe.sendMessage(
PresenceMessages.protocolRequest("Italy",
"null", "Bittorrent", ipDir, code));

stdOut.append("Request Message Sent \n");
System.out.println("Message Request Sent");
}
} catch (IOException e1) {
stdOut.append("AcceptError \n");
System.out.println("Accept Error");
}

/*
 * In the meanwhile we create an Input Propagate Pipe to see if we catch
 * another user searching a peer
 */

inPipe = pipeService.createInputPipe(propagatePipeAdv, mHandler);

if (inPipe != null) {
stdOut.append("Input Pipe Created \n");
System.out.println("InputPipe Created");
}

try {
```

```
stdOut.append("Sleeping Test \n");
System.out.println("Sleeping test");
Thread.sleep(10000);

} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
//OJO CHEQUEAR Y ENVIAR UN MSGPIPELISTENER EVENT
/*try {
inPipe.waitForMessage();
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}*/

/*while (true) {

}*/

} catch (IOException e) {
e.printStackTrace();
}

}

return 0;
}

public InputPipe getInPipe() {

return inPipe;
}

public void stopApp() {
//Close pipes when stopping the service
inPipe.close();
//outPipe.close();
}

public void sendMessage() {
// TODO Auto-generated method stub

}

}
```

Appendix F

Neutrality Service: Code extract

```
package jxta.services.presence.messagehandler;

import gooee.NeubotGui;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.URL;
import java.nio.channels.FileChannel;
import java.util.Random;

import javax.swing.JScrollBar;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

import org.klomp.snark.Snark;

import statstorage.StoreTable;

import compare.CaptureComparison;

//import capture.Capture11;
import capture.Capturer;
//import java.net.Inet4Address;

//import org.klomp.snark.Snark;

import jpcap.JpcapCaptor;
import jpcap.NetworkInterface;
import jpcap.NetworkInterfaceAddress;
import jxta.protocols.bittorrent.BitTorrentRunnable;
import jxta.services.filexchange.SocketClient;
import jxta.services.filexchange.SocketServer;
```

```
import jxta.services.presence.PresenceMessages;
import jxta.servicetest.FirstServiceImpl;
import net.jxta.document.AdvertisementFactory;
//import net.jxta.document.Document;
import net.jxta.document.StructuredDocumentFactory;
import net.jxta.document.XMLDocument;
import net.jxta.endpoint.Message;
import net.jxta.endpoint.MessageElement;
import net.jxta.endpoint.Message.ElementIterator;
import net.jxta.id.IDFactory;
import net.jxta.peer.PeerID;
import net.jxta.peergroup.PeerGroup;
import net.jxta.pipe.PipeID;
import net.jxta.pipe.PipeMsgEvent;
import net.jxta.pipe.PipeMsgListener;
import net.jxta.protocol.PipeAdvertisement;
import net.jxta.util.JxtaBiDiPipe;

public class MessageHandler implements PipeMsgListener {

private PeerGroup neubotGroup = null;

private PipeID propagatePipeID = null;

private FirstServiceImpl serviceHandler = null;

private JxtaBiDiPipe biDiPipe = null;

private Snark bitTorrentC, bitTorrentT;

private Capturer capture;

private Thread t;

private String ipDir = null;

private String remoteIpDir = "";

private Thread t_capture;

private boolean done_peer = false, done_tracker = false;

private JTextArea stdOut;

private JScrollPane stdOutScroll;

private int sequenceCode = -1;

private BitTorrentStats stats = new BitTorrentStats();

ClassLoader cl = this.getClass().getClassLoader();

public MessageHandler(FirstServiceImpl serviceHandler,
PipeID propagatePipeID, PipeID bidiPipeID, String ipDir,
JTextArea stdOut, JScrollPane stdOutScroll) {

this.serviceHandler = serviceHandler;

this.propagatePipeID = propagatePipeID;

this.neubotGroup = serviceHandler.getNeubotGroup();
```

```
this.ipDir = ipDir;

this.stdOut = stdOut;

this.stdOutScroll = stdOutScroll;

this.stats.setLocalIp(ipDir);

}

public void pipeMsgEvent(PipeMsgEvent evt) {
// TODO Auto-generated method stub

Message message = evt.getMessage();

PipeID receivedPipeID = evt.getPipeID();

if (receivedPipeID.equals(propagatePipeID) &&
serviceHandler.getPeerState().equalsIgnoreCase("IDDLE")) {

NeubotGui.appendText(stdOutScroll, stdOut,
message.getMessageElement("NeubotPresence",
"PipeAdv").toString() + "\n");
/*stdOut.append(message.getMessageElement("NeubotPresence",
"PipeAdv").toString() + "\n");*/
System.out.println(message.getMessageElement("NeubotPresence",
"PipeAdv"));

try {
XMLDocument<?> pipeAdvDoc = (XMLDocument<?>)
StructuredDocumentFactory.newStructuredDocument(
message.getMessageElement("NeubotPresence",
"PipeAdv"));

PipeAdvertisement pipeAdv = (PipeAdvertisement)
AdvertisementFactory.newAdvertisement(pipeAdvDoc);

try {

biDiPipe  = new JxtaBiDiPipe();

biDiPipe.setMessageListener( this );

biDiPipe.setReliable( true );

try {

PeerID senderID = (PeerID) IDFactory.fromURI(
new URI(message.getMessageElement(
"NeubotPresence","SenderPeerID").toString()));

NeubotGui.appendText(stdOutScroll, stdOut,
"Connecting to Server BiDiPipe \n");
//stdOut.append("Connecting to Server BiDiPipe \n");
System.out.println("Connecting to Server biDiPipe");

biDiPipe.connect(neubotGroup, senderID, pipeAdv, 5000, this);

NeubotGui.appendText(stdOutScroll, stdOut,
"Connected to Server BiDiPipe \n");
//stdOut.append("Connected to Server BiDiPipe \n");
System.out.println("Connected to Server biDiPipe!");
```

```
serviceHandler.setPeerState("PAIRED");

NeubotGui.appendText(stdOutScroll, stdOut, "Peer State: " +
serviceHandler.getPeerState() + "\n");
//stdOut.append("Peer State: " + serviceHandler.getPeerState() + "\n");
System.out.println(serviceHandler.getPeerState());

} catch (URISyntaxException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}

} catch (IOException e) {

NeubotGui.appendText(stdOutScroll, stdOut,
"Failed Connection to Server BiDiPipe \n");
//stdOut.append("Failed Connection to Server BiDiPipe \n");
System.out.println("Failed Connection to Server BiDiPipe");

try {

PeerID senderID = (PeerID) IDFactory.fromURI(
new URI(message.getMessageElement(
"NeubotPresence","SenderPeerID").toString()));

biDiPipe.connect(neubotGroup, senderID, pipeAdv, 5000, this);

serviceHandler.setPeerState("PAIRED");
NeubotGui.appendText(stdOutScroll, stdOut,
"Connected to Server BiDiPipe \n");
//stdOut.append("Connected to Server BiDiPipe \n");
System.out.println("Connected to Server biDiPipe!");

} catch (URISyntaxException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}

}
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

} else {

if (biDiPipe == null)
biDiPipe = serviceHandler.getBiDiPipe();

try {
if (biDiPipe.getInputPipe().getPipeID().equals(evt.
getPipeID())){

MessageElement messageType = message.
getMessageElement("NeubotPresence",
"MessageType");
if (!(messageType == null)) {
if(messageType.toString().equalsIgnoreCase(
"Request") &&
serviceHandler.getPeerState().equalsIgnoreCase("PAIRED")) {
```

```
NeubotGui.appendText(stdOutScroll, stdOut,
"Received a Request Message from the BiDiPipe \n");
//stdOut.append("Received a Request Message from the BiDiPipe \n");
System.out.println("Received a Request Message form" +
" the BiDiPipe");

MessageElement sequenceElement = null;

sequenceElement =
message.getMessageElement("NeubotPresence",
"TransactionCode");

NeubotGui.appendText(stdOutScroll, stdOut, "RequestCode: " +
sequenceElement.toString() + "\n");
//stdOut.append("RequestCode: " + sequenceElement.toString() + "\n");
System.out.println("RequestCode: " + sequenceElement.toString());

ElementIterator messageElements =
message.getMessageElementsOfNamespace(
"NeubotPresence");

//HERE WE SHOULD PROCESS THE REQUEST

try {

if (sequenceElement != null) {

//String sequenceCode = sequenceElement.toString();

this.sequenceCode = Integer.parseInt(sequenceElement.toString());

stats.setTransactionID(sequenceCode);

NeubotGui.appendText(stdOutScroll, stdOut,
"Sending Protocol Response \n");
//stdOut.append("Sending Protocol Response \n");
System.out.println("Sending Protocol Response");

remoteIpDir = message.getMessageElement(
"NeubotPresence", "IP").toString();

stats.setRemoteIp(remoteIpDir);

biDiPipe.sendMessage(PresenceMessages.protocolAvailable(
sequenceCode, "Italy", "null", "Bittorrent", "true", ipDir));

NeubotGui.appendText(stdOutScroll, stdOut,
"Protocol Response Sent \n");
//stdOut.append("Protocol Response Sent \n");
System.out.println("Protocol Response Sent");

} else {
NeubotGui.appendText(stdOutScroll, stdOut,
"Request Message received without Sequence Code \n");
System.out.println("Request Message received" +
" without sequence code (do nothing)");
}

} catch (IOException e) {

e.printStackTrace();
}
```

```
while (messageElements.hasNext()) {
NeubotGui.appendText(stdOutScroll, stdOut,
messageElements.next().toString() + "\n");
//stdOut.append(messageElements.next().toString() + "\n");
System.out.println(messageElements.next());

}

} else if (messageType.toString().equalsIgnoreCase("Response") &&
Integer.parseInt(message.getMessageElement("NeubotPresence",
"TransactionCode").toString()) == sequenceCode) {

NeubotGui.appendText(stdOutScroll, stdOut, "Response Received \n");
//stdOut.append("Response Received \n");
System.out.println("Response Received!");

//ElementIterator messageElements =
// message.getMessageElementsOfNamespace(
// "NeubotPresence");

//MessageElement sequenceElement =

// message.getMessageElement(
// "NeubotPressence", "ResponseCode");

/*while (messageElements.hasNext()) {

System.out.println(messageElements.next());
}*/

NeubotGui.appendText(stdOutScroll, stdOut, "Starting BitTorrent
Tracker \n");
//stdOut.append("Starting BitTorrent Tracker \n");
System.out.println("Starting Bittorrent tracker");

remoteIpDir = message.getMessageElement("NeubotPresence",
"IP").toString();

/*BitTorrentRunnable bitTorrent = new BitTorrentRunnable("tracker");

bitTorrent.run();*/
//File f = new File("random1500x1500.jpg");


//File f = new File("mac.pdf");

File f1 = new File("sent_" + ipDir);

//if (f1.createNewFile()) System.out.println("New file created");
if (f1 != null) System.out.println("File sent created");

/*if (f.renameTo(f1)){
NeubotGui.appendText(stdOutScroll, stdOut, "FileName Changed \n");
//stdOut.append("FileName Changed \n");
System.out.println("FileName changed");
} else {
NeubotGui.appendText(stdOutScroll, stdOut, "FileName Change Failed \n");
//stdOut.append("FileName Changed \n");
System.out.println("change name failed");
}*/
```

```
f1.deleteOnExit();

//File f2 = new File(cl.getResource("mac.pdf").getFile());

URL inUrl = this.getClass().getClassLoader().getResource(
"random1500x1500.jpg");
System.out.println(inUrl);
InputStream in = inUrl.openStream();
if (in != null) System.out.println("archivo leido");
    OutputStream out = new FileOutputStream(f1);

        // Transfer bytes from in to out
        byte[] buf = new byte[1024];
        int len;
        while ((len = in.read(buf)) > 0) {
            out.write(buf, 0, len);
        }
        in.close();
        out.close();


String[] params = new String[5];

params[0] = "--port";

params[1] = "12345";

params[2] = "--share";

params[3] = ipDir;

params[4] = "sent_" + ipDir;

bitTorrentT = new Snark(params, true, stats);

t = new Thread(bitTorrentT);

t.start();

done_tracker = true;

String trackerURL = BitTorrentRunnable.getTrackerURL();

Message protocolInit = PresenceMessages.protocolInit(
sequenceCode, "Bittorrent", "tracker");
//System.out.println( protocolInit );
biDiPipe.sendMessage(protocolInit);

NeubotGui.appendText(stdOutScroll, stdOut,
"Sending BitTorrent URL Tracker \n");
//stdOut.append("Sending BitTorrent URL Tracker \n");
System.out.println("Sending Bittorrent URL tracker");

Message protocolStarted = PresenceMessages.protocolStarted(
sequenceCode, "Bittorrent", trackerURL);

//ElementIterator ite = protocolStarted.getMessageElements();

/*while ( ite.hasNext() ) {

MessageElement el = ite.next();
System.out.println( "Element: " + ite.getNamespace() + " :: " +
```

```
el.getElementName());
System.out.println( "[" + el + "]" );
}*/

biDiPipe.sendMessage(protocolStarted);

NeubotGui.appendText(stdOutScroll, stdOut, "Patameters Sent \n");
//stdOut.append("Patameters Sent \n");
System.out.println("Parameters sent");

NetworkInterface[] devices = JpcapCaptor.getDeviceList();

int interf = 0;

for (int i = 0; i < devices.length; i++) {

NetworkInterfaceAddress addresses[] = devices[i].addresses;

for (int j = 0; j < addresses.length; j++) {

if (addresses[j].address.toString().substring(1).equalsIgnoreCase(
ipDir))
interf = i;
System.out.println(i + " " + addresses[j].address.toString());
}
}
//int i = 0;
/*while(i < devices.length) {

System.out.println(devices[i].name);
i++;

}*/
System.out.println("local interface " + devices[interf]);
System.out.println("local IP " + ipDir);
System.out.println("remote IP " + remoteIpDir );
System.out.println(interf);

//Thread.sleep(1000);

capture = new Capturer(devices[interf], "capture_tracker");

capture.setIPCaptureFilter(remoteIpDir, ipDir);

t_capture = new Thread(capture);

t_capture.start();

} else if (messageType.toString().equalsIgnoreCase(
"ProtocolInitialization") && Integer.parseInt(
message.getMessageElement("NeubotPresence",
"TransactionCode").toString()) == sequenceCode) {

NeubotGui.appendText(stdOutScroll, stdOut,
"Waiting for Remote Parameters from Tracker \n");
//stdOut.append("Waiting for Remote Parameters from Tracker \n");
System.out.println("Waiting for remote parameters from Tracker");

} else if (messageType.toString().equalsIgnoreCase("ProtocolStarted")
&& Integer.parseInt(message.getMessageElement("NeubotPresence",
"TransactionCode").toString()) == sequenceCode) {
```

```
NetworkInterface[] devices = JpcapCaptor.getDeviceList();

int interf = 0;

for (int i = 0; i < devices.length; i++) {

NetworkInterfaceAddress addresses[] = devices[i].addresses;

for (int j = 0; j < addresses.length; j++) {

if (addresses[j].address.toString().substring(1).equalsIgnoreCase(
ipDir))
interf = i;
System.out.println(i + " " + addresses[j].address.toString());

}
}


String[] params = new String[1];

params[0] = message.getMessageElement("TrackerURL").toString();

capture = new Capturer(devices[interf], "capture_client");

capture.setIPCaptureFilter( params[0].substring(7,
params[0].indexOf(":12345/metainfo.torrent")), ipDir );

t_capture = new Thread( capture );

t_capture.start();

try {
Thread.sleep(1000);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

NeubotGui.appendText(stdOutScroll, stdOut, "Initializing BitTorrent Client \n");
//stdOut.append("Initializing BitTorrent Client \n");
System.out.println("Initializing Bittorrent client");

//(new Thread(new BitTorrentRunnable())).start();

bitTorrentC = new Snark(params, false, stats);

t = new Thread(bitTorrentC);
t.start();

done_peer = true;

//System.out.println(bitTorrentC.getIsTracker());

try {
t.join();
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
t = null;
```

```
/*stats.setDownload(bitTorrentT.stats.download);
stats.setUpload(bitTorrentT.stats.upload);*/

bitTorrentC = null;

NeubotGui.appendText(stdOutScroll, stdOut, "Returned from Snark \n");
//stdOut.append("Returned from Snark \n");
System.out.println("Returned from Snark");

Message messageToSend = PresenceMessages.protocolFinishing(
sequenceCode, "BitTorrent", String.valueOf(stats.isDownload()));

biDiPipe.sendMessage(messageToSend);

//JpcapWriter writer = capture.getFileWriter();
try {
Thread.sleep(1000);
} catch (InterruptedException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}


capture.setCloseCaptor();

try {
t_capture.join(1000);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

//writer.close();

NeubotGui.appendText(stdOutScroll, stdOut, "Capture Stopped \n");
//stdOut.append("Capture Stopped \n");
System.out.println("Capture Stopped");
/*BitTorrentRunnable bitTorrent = new BitTorrentRunnable(
message.getMessageElement("TrackerURL").toString());

bitTorrent.run();*/

} else if (messageType.toString().equalsIgnoreCase(
"ProtocolFinishing") && Integer.parseInt(message.getMessageElement(
"NeubotPresence", "TransactionCode").toString()) ==
sequenceCode) {

bitTorrentT.setFinish();

try {
t.join();
} catch (InterruptedException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}
t = null;

bitTorrentT = null;

capture.setCloseCaptor();
```

```
try {
t_capture.join(1000);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

NeubotGui.appendText(stdOutScroll, stdOut, "Capture Stopped \n");
//stdOut.append("Capture Stopped \n");
System.out.println("Capture Stopped");

Message messageToSend = PresenceMessages.protocolFinished(
sequenceCode, "BitTorrent");

//ElementIterator ite = messageToSend.getMessageElements();

/*while ( ite.hasNext() ) {

MessageElement el = ite.next();
System.out.println( "Element: " + ite.getNamespace() + " :: " +
el.getElementName());
System.out.println( "[" + el + "]" );
}*/

biDiPipe.sendMessage(messageToSend);

NeubotGui.appendText(stdOutScroll, stdOut,
"remote Peer (Client) finishing BitTorrent \n");
//stdOut.append("remote Peer (Client) finishing BitTorrent \n");
System.out.println("Remote peer (client) finishing bitTorrent");

//writer.close();

NeubotGui.appendText(stdOutScroll, stdOut, "Capture Stopped \n");
//stdOut.append("Capture Stopped \n");
System.out.println("Capture Stopped");


}

else if (messageType.toString().equalsIgnoreCase("ProtocolFinished")
&& Integer.parseInt(message.getMessageElement("NeubotPresence",
"TransactionCode").toString()) == sequenceCode) {

//bitTorrent.setFinish();


NeubotGui.appendText(stdOutScroll, stdOut,
"Remote Peer (Tracker) finishing BitTorrent \n");
//stdOut.append("Remote Peer (Tracker) finishing BitTorrent \n");
System.out.println("Remote peer (tracker) finishing bitTorrent");

try {
Thread.sleep(1000);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}


if ( !( done_peer && done_tracker ) ) {
```

```
//generate Request message

Random randomGenerator = new Random();
sequenceCode = randomGenerator.nextInt(65535);

Message toSend = PresenceMessages.protocolRequest("Italy",
"null", "Bittorrent", ipDir, sequenceCode);
biDiPipe.sendMessage(toSend);
//done = true;

NeubotGui.appendText(stdOutScroll, stdOut,
"Sending a new Request \n");
//stdOut.append("Sending a new Request \n");
System.out.println("Sending a new Request");
} else {

SocketServer ssocket = new SocketServer(neubotGroup);

String socketID = ssocket.getSocketID();

Message toSend = PresenceMessages.protocolExStart(sequenceCode,
"Nada", socketID);
biDiPipe.sendMessage(toSend);

//Start Server Socket

ssocket.run();

CaptureComparison compare = new CaptureComparison(stdOut,
stdOutScroll);
try {
compare.compare("capture_client", "capture_tracker_remote");
stats.setLcpl(compare.getLpl());
stats.setRtpl(compare.getRpl());


} catch (ClassNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
try {

compare.compare("capture_tracker", "capture_client_remote");
stats.setLtpl(compare.getLpl());
stats.setRcpl(compare.getRpl());

} catch (ClassNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
//ssocket = null;
//Return to IDDLE State
//biDiPipe.close();
done_peer = false;
done_tracker = false;
/*ssocket = null;
```

```
csocket = null;*/

serviceHandler.setPeerState("IDDLE");
File f1 = new File("sent_" + remoteIpDir);
f1.delete();

StoreTable store = new StoreTable();
store.insertBitTorrentStats(stats);
}
/* else {
}

SocketServer server = new SocketServer(neubotGroup);
server.run();
}*/



/*try {
t.join();
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}*/

//System.out.println("Regrese de Capture Thread");
}

else if (messageType.toString().equalsIgnoreCase(
"SocketServerStarted") && Integer.parseInt(
message.getMessageElement("NeubotPresence",
"TransactionCode").toString()) ==
sequenceCode) {

System.out.println("Socket Client must start here");
String socketID = message.getMessageElement("NeubotPresence",
"SocketID").toString();
SocketClient csocket = new SocketClient(neubotGroup, socketID);
csocket.run();
CaptureComparison compare = new CaptureComparison(stdOut,
stdOutScroll);
try {
compare.compare("capture_client", "capture_tracker_remote");
stats.setLcpl(compare.getLpl());
stats.setRtpl(compare.getRpl());

} catch (ClassNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

try {
compare.compare("capture_tracker", "capture_client_remote");
stats.setLtpl(compare.getLpl());
stats.setRcpl(compare.getRpl());
if (compare.isSwitched()) {
System.out.println(compare.getNotFound() +
" packets were captured by the" +
" remote peer but were not captured by the local peer");
```

```
} else {
System.out.println(compare.getNotFound() +
" packets were captured by the" +
" local peer but were not captured by the remote peer");
}

} catch (ClassNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
//Return to IDDLE State
done_peer = false;
done_tracker = false;
//biDiPipe.close();
/*ssocket = null;
csocket = null;*/
serviceHandler.setPeerState("IDDLE");
File f1 = new File("sent_" + remoteIpDir);
f1.delete();

StoreTable store = new StoreTable();
store.insertBitTorrentStats(stats);


} //Supongo que else! y da capo
}
else {

NeubotGui.appendText(stdOutScroll, stdOut,
"Message Arrived from an unknown pipe it is going to be discarded \n");
System.out.println("Message Arrived from an unknown " +
"pipe is going to be discarded");

//Temporarily

}
}

} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
}

public void setSequenceCode(int code) {

this.sequenceCode = code;
}


public class BitTorrentStats {

private String localIp, remoteIp;
private int transactionID, ltpl, rtpl, lcpl, rcpl;
private long downloadedBytes, uploadedBytes;
private boolean download, upload;

public BitTorrentStats () {
```

```
localIp = null; remoteIp = null;
transactionID = 0; ltpl = 0; rtpl = 0; rcpl = 0; lcpl = 0;
download = false;
upload = false;
}
public String getLocalIp() {
return localIp;
}
public String getRemoteIp() {
return remoteIp;
}
public void setLocalIp(String ip) {
this.localIp = ip;
}
public void setRemoteIp(String ip) {
this.remoteIp = ip;
}
public int getTransactionID() {
return transactionID;
}
public void setTransactionID(int transactionID) {
this.transactionID = transactionID;
}
public boolean isDownload() {
return download;
}
public void setDownload(boolean download) {
this.download = download;
}
public boolean isUpload() {
return upload;
}
public void setUpload(boolean upload) {
this.upload = upload;
}
public long getDownloadedBytes() {
return downloadedBytes;
}
public void setDownloadedBytes(long downloadedBytes) {
this.downloadedBytes = downloadedBytes;
}
public long getUploadedBytes() {
return uploadedBytes;
}
public void setUploadedBytes(long uploadedBytes) {
this.uploadedBytes = uploadedBytes;
}
public int getLtpl() {
return ltpl;
}
public void setLtpl(int ltpl) {
this.ltpl = ltpl;
}
public int getRtpl() {
return rtpl;
}
public void setRtpl(int rtpl) {
this.rtpl = rtpl;
}
public int getLcpl() {
return lcpl;
}
```

```
public void setLcpl(int lcpl) {
this.lcpl = lcpl;
}
public int getRcpl() {
return rcpl;
}
public void setRcpl(int rcpl) {
this.rcpl = rcpl;
}

}
}
```

Bibliography

[1] Angele A. Gilroy *Net Neutrality: Background and Issues* Library of Congress Washington DC Congressional Research Service, 2006.

[2] Eric Schmidt *A Note to Google Users on Net Neutrality* `http://www.google.com/help/netneutrality_letter.html`

[3] Chris Metz *Interconnecting ISP Networks* IEEE Internet Computing, Volume 5, Issue 2, March 2001.

[4] Russ Haynal *Internet: "The Big Picture"* `http://navigators.com/internet_architecture.html`

[5] Jon Crowcroft *Net Neutrality: The Technical Side of the Debate A White Paper* International Journal of Communication, Vol 1. 2007.

[6] Tim Wu *Network Neutrality, Broadband Discrimination* Journal of Telecommunications and High Technology Law, Vol. 2, p. 141, 2003.

[7] Nate Anderson *Deep packet inspection meets Net neutrality* CALEA, 2007.

[8] *Procera Product Specifications* `http://www.proceranetworks.com/products/packetlogic-hardware-platforms.html`

[9] Scott Jordan *A Layered Network Approach to Net Neutrality* International Journal of Communication Vol. 1, 2007.

[10] *Save The Internet FAQ* `http://www.savetheinternet.com/=faq`

[11] Daniel J. Weitzner *The Neutral Internet: An Information Architecture for Open Societies*, MIT Computer Science and Artificial Intelligence Laboratory, `http://dig.csail.mit.edu/2006/06/neutralnet.html`

[12] Peter Svensson *Comcast Blocks Some Internet Traffic* Associater Press, `http://www.sfgate.com/cgi-bin/article.cgi?f=/n/a/2007/10/19/financial/f061526D54.DTL&feed=rss.business`, 2007.

[13] M. Dischinger, A. Haeberlen, K. P. Gummadi, S. Saroiu *Characterizing Residential Broadband Networks* Proc. Internet Measurement Conference, 2007.

[14] Xiaowei Tang, Gene Tsudik, Xin Liu *A Technical Approach to Net Neutrality* `www.ics.uci.edu/~gts/paps/ylt06.pdf`

[15] Dana Moore, John Hebeler *Peer-to-Peer: Building Secure, Scalable, and Manageable Networks* Mc Graw Hill, 2002.

[16] Mark Wielaard *The Hunting of the Snark Project - BitTorrent Application Suite* http://www.klomp.org/snark/

[17] Jeffrey D. Schank *Il Manuale Client Server* Mc Graw Hill - Italia, 1994.

[18] Andy Oram *Peer-to-Peer: Harnessing the Power of Disruptive Technologies* O'Reilly, 2001.

[19] Brendon J. Wilson *JXTA* New Riders, 2002.

[20] Thomas Porter *The Perils of Deep Packet Inspection* http://www.securityfocus.com/infocus/1817, 2001.

[21] Tom White *Scheduling recurring tasks in Java applications* http://www.ibm.com/developerworks/web/library/j-schedule.html, 2003.

[22] IETF *RFC 3148* http://www.ietf.org/rfc/rfc3148.txt, 2001.

[23] Scott Oaks, Bernard Traversat *Getting Started with JXTA* O'Reilly, http://www.onjava.com/pub/a/onjava/excerpt/jxtaian_2/index1.html

[24] Matthew Lehman *Measuring throughput in a TCP/IP network* TechRepublic, http://articles.techrepublic.com.com/5100-10878_11-1042973.html, 2001.

[25] Mike Duigou *Discovery to Exhaustion : JXTA* Weblog of Mike Duigou, http://blogs.sun.com/bondolo/

[26] *JXSE Programmers Guide v2.5* https://jxta-guide.dev.java.net/source/browse/jxta-guide/trunk/src/guide_v2.5/

[27] Alan Beecraft *Peer-to-Peer: From JMS to Project JXTA Part 1: Shall we chat?* Sun Developer Network, http://java.sun.com/developer/technicalArticles/peer/ , 2001.

[28] *JXTA Community Projects* https://jxta.dev.java.net.

[29] Ian J. Taylor *From P2P to Web Services and Grids: Peers in a Client/Server World* Springer, 2005.

[30] *Java Web Start Developer's Guide* http://java.sun.com/products/javawebstart/1.2/docs/developersguide.html#dev