Master's thesis
Design of the Session Layer Gateway Supporting Mobile, Delay and Disconnection Tolerant Communication

Daniel Abramowicz
(dabr@kth.se)

March 27, 2007

Abstract

Today mobility within Internet is rather regarded as a problem instead of as a fact. The term mobility is also somewhat dubious with no standard definition. Mobility is, in our view, not just the physical mobility of a laptop but even the move of a directory from one computer to another. Mobility has evolved but Internet has not followed in the same pace.

To deal with the lack of mobility in Internet we introduce a gateway supporting a new kind of entity, endpoints. An endpoint can be a user, content, an application and they have their own naming system. With a name server allowing real-time update of ip addresses of endpoints these endpoints can be moved from one computer in one network to another computer in another network. It is here that a *session layer gateway* is needed. Not all networks are of the same address scheme, there are both IPv4 networks and IPv6 networks today, and there are private IPv4 networks using Network Address Translation. The communication between these networks need to be bridged in order to extend mobility beyond the limitations of a specific network type.

The session layer gateway is in our design part of the network infrastructure. By this design we can ensure simultaneous mobility of endpoints, two endpoints communicating can simultaneously move their current point of attachment to other points of attachment and then resume their suspended communication session.

A session layer gateway has been designed and implemented to show that this is feasible. With our design it is not a problem changing network during an ongoing communication session and later resume it on a network of a different addressing scheme, it is a fact.

Contents

List of Tables

List of Figures

1 Introduction

Internet is well functioning for the purpose which it was built, for desktops computers steadfast in one place. Today laptops are common and people do not only bring them with themselves from one place to another but also move around in the office with their laptops expecting full networking connectivity. This kind of mobility has not been developed to the same extent for Internet itself as for the physical mobility of laptops. Internet has since it started relied on static mappings between all of its components. Users, computers, networks and links were not expected to be mobile - nowadays they are.

The definition of mobility is not clear. Depending on what is concerned the definition differs. The mobile phone implies that you can move around with it, even travel with it in a car or on a train and still keep your call ongoing. However, if you loose your connection there is no way to resume your phone call, you simply need to re-dial.

With respect to computers the definition of mobility might also vary. It used to be only the possibility to physically bring your laptop with you. You shutdown your computer in one place and booted it at another place. If you had any previous connections they were lost and the other end did not know where to keep sending packets. Approaches to this problem have been proposed such as Mobile IP.

Our definition of mobility concerns the change of network address and also location while still keeping all connections. We also extend our definition of mobile objects to not only refer to computers but also to users, content and applications.

To achieve this level of mobility where the connection can be lost, willingly or unwillingly, we need a way to suspend and resume connections, in [5] definitions of suspend and resume are given. Which are used in this report.

With this concept of mobility, transparent Internetworking is an important issue. To suspend in one place and then move and resume in another place with a different protocol or addressing scheme requires some kind of translator mechanism, before you can move freely between for example IPv4 and IPv6 networks.

The remainder of this thesis is structured as follows. Chapter 2 presents the background including the problem statement and motivation and also explores some related work. The design part, concepts and ideas put together, is given in chapter 3. Chapter 4 starts with an overview of the implementation and is followed by the implementation in depth. In chapter 5 the analysis with results is presented. Finally, some conclusions and future work is given in chapter six.

2 Background

In this part of the thesis a background to networking and the problems the Internet of today faces will be given. It will shortly describe some of the different Internet mobility approaches, with their benefits and drawbacks, and argue for why our approach is the best suited.

2.1 Networking concepts

2.1.1 The OSI model

The OSI model [16] is a seven layer model design of network systems allowing communication between different computer systems. The model was introduced in the late 1970s and is used to schematically display network communication.

The idea behind the layered model is that each layer only provides information for the layer above it and only uses information from the layer below it, on a single machine. When communicating between machines layers will only talk to its corresponding layer on the other machine.

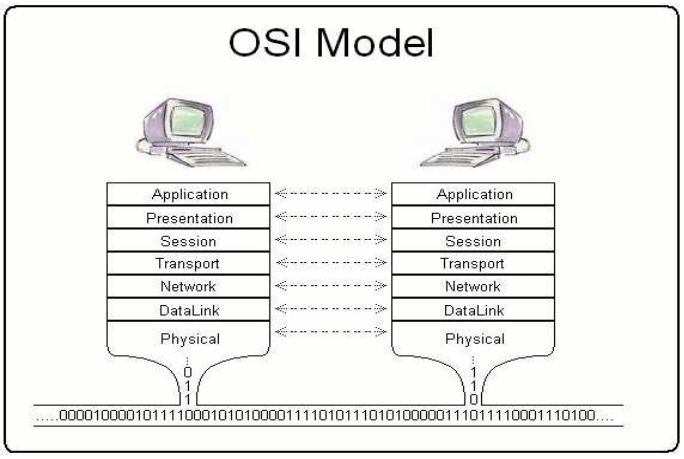

Figure 1: The OSI Layer Model displaying the end-to-end communication between layers on different machines. Picture under GNU Free Documentation License from http://upload.wikimedia.org/wikipedia/en/f/ff/Osi_model_trad.jpg

Application Layer The top layer of the OSI model provides access to the network resources through applications which are used by humans.

Presentation Layer The presentation layer provides translation, encryption and compressing of data. The translation mechanism communicates with the other peer to ensure that data transferred uses the same encoding system. For example, bit streams can be either of big or little endian.

Session Layer The session layer provides the means to establish, manage and terminate sessions. These are the services it is supposed to provide.

Session-connection establishment
: This service enables two presentation entities to establish communication through the session layer. The presentation entities are identified by their session address.

Session-connection release
: Allows the release of a connection without any loss of data.

Normal data transfer
Allows sending data between presentation-entities.

Expedited data transfer
Allows sending data with special priority to bypass the ordinary data stream.

Token management
Gives the presentation entities the ability to control whose turn it is to perform certain control functions.

Session-connection synchronisation
Identifies synchronisation points on where to resume a connection upon a disconnection.

Checkpointing Checkpointing is a way to ensure that data is received correctly by establishing mutually agreed checkpoints from where to recover data if needed. The session layer is not responsible for any checkpointing. However, the OSI Model recognises the need of checkpointing and recovery control but leaves it for applications to deal with, for example file transfer.

Exception reporting
Permits the presentation entities to be notified of exceptional situations.

Activity management
Allows users to identify logical pieces of work as activities. One activity per session connection is allowed, but a session connection can have many consecutive activities. Activities can be interrupted and then resumed on the same connection or a subsequent one.

Typed data transfer
Allows a presentation entity to send a session-service-data-unit to another presentation entity regardless of token management.

Resynchronisation
Resynchronisation allows two entities to re-setup a connection with new tokens and the synchronisation point to a new value. A resynchronisation may purge undelivered data.

Transport Layer The transport layer's obligations are to provide reliable end-to-end message delivery and error recovery on a process-to-process basis. It makes sure that a message sent from one machine arrives safely at its destination.

Connection mode services

Transport-connection establishment
This service allows two session entities to establish a connection identified by their session addresses.

Transport-connection release
When a session entity releases the transport connection the other session entity is informed.

Data transfer
Allows sending data between session entities.

Expedited data transfer
Allows sending data with special priority to bypass the ordinary data stream.

Suspend facility
The facility to suspend a connection between two session entities.

Connectionless services When sending over connectionless mode each packet must contain all the information needed to send it to its destination, which allows independent delivery of packets. It does not provide segmentation and reassembly of packets, nor does it give guarantees for delivery or correct ordering of data.

Network Layer The network layer is responsible for the source-to-destination delivery of packets, ie delivering data between transport entities. It is also responsible for routing the data to its destination based on the network address.

Data Link Layer The data link layer is responsible for transferring data between network entities. These network entities are identified by their data link address. The layer also have an error detection and correction mechanism. The data link layer divides the stream of bytes received by the network layer into smaller data units called frames.

Physical Layer The physical layer is the actual physical media which connects machines. On this layer bits are transferred between machines.

2.1.2 The TCP/IP-stack

The OSI model is a theoretical description of the network environment. However, its usefulness is subject to debate among developers [13].

In practice the TCP/IP-stack is used. The TCP/IP-stack is similar to the OSI-model as can be seen in the figure below.

Application Layer An example of a protocol in the application layer is the Hypertext Transfer Protocol (HTTP) [22], a common text based protocol used for webtransfer. The TCP/IP stack supports many protocols in this layer.

Transport Layer The protocols defined by the TCP/IP stack in the transport layer are the Transmission Control Protocol (TCP) [15] and the User Datagram Protocol (UDP) [17]. Both UDP and TCP uses port numbers as address identifiers of the services to connect to on a specific host.

UDP is a connectionless protocol which does not guarantee that the datagram reaches its destination. An example of an application on top of UDP is Voice over IP.

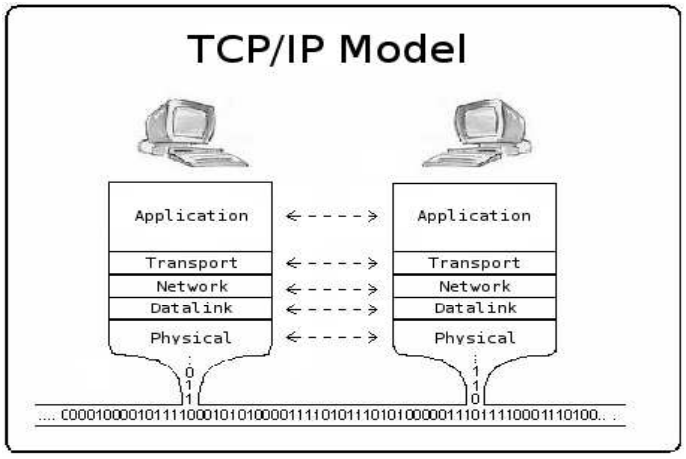

Figure 2: The TCP/IP stack model. A modified version of the picture under GNU Free Documentation License from http://upload.wikimedia.org/wikipedia/en/f/ff/Osi_model_trad.jpg

TCP is a connection oriented protocol which sets up a connection between the hosts before sending TCP packets between them. An example of an application on top of TCP is file transferring using the File Transfer Protocol.

Additional protocols are Stream Control Transmission Protocol (SCTP) [27] and Datagram Congestion Control Protocol (DCCP) [10]. SCTP provides reliable and in-order data delivery like TCP but is message-oriented. It also adds some features such as multi-streaming and graceful shutdown. DCCP is similar to UDP, it is an unreliable packet stream protocol but has the additional feature of congestion control. This is useful for Voice over IP applications who otherwise would either use TCP or implement their own congestion control in the application layer.

Network Layer The protocol supported by the TCP/IP stack in the network layer is the Internet Protocol (IP). There are two versions of IP in use, IPv4 [14] and IPv6 [28].

IPv4 IP version 4 is the most widely used network protocol in the Internet today. It is a best effort control protocol and does not guarantee delivery.

Address and address space IPv4 uses 32-bit addresses, logical addresses, to identify hosts uniquely in a network. With 32 bit addresses the address space is 2^{32} which is more than 4 billion addresses. This can be extended with techniques such as Network Address Translation (NAT) where a router is assigned an address on the outer side and on the inner side hands out addresses defined as private [32] to its own hosts.

ARP and RARP Packets need to pass through physical networks to reach its destination. In the physical layer hosts and routers are ad-

dressed with their physical address. There is need for a mapping between physical and logical addresses as the delivery of packets requires two levels of addresses. Two protocols are defined: the Address Resolution Protocol (ARP) [8] and the Reverse Address Resolution Protocol (RARP) [23].

ICMP The Internet Control Message Protocol (ICMP) [18] is not a protocol on its own but is a part of IP. IP is not reliable and ICMP does not provide reliability, it only adds the possibility to receive feedback of problems in the communication. An example when ICMP is used is when a datagram cannot reach its destination.

IGMP The Internet Group Management Protocol (IGMP) [30] is also a part of IP. It is used to report multicast group memberships to routers.

IPv6 IP version 6 is the successor to IPv4.

Address and address space One of the issues with IPv4 is that the address space is too small. IPv6 uses 128-bit addresses which increases the address space to 2^{128}.

ARP and RARP The ARP protocol is incorporated into ICMPv6. RARP was not used often in IPv4 and therefore there is no RARP in IPv6.

ICMPv6 ICMPv6 [3] is as ICMP used to report errors such as destination unreachable but has also been given the functions of ARP and IGMP.

IGMP In IPv6 this feature is incorporated into ICMPv6.

Data Link Layer A Media Access Control (MAC) protocol, for example the IEEE 802.3 standards which defines the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol and IEEE 802.11b which defines the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol. The TCP/IP stack does not define any specific protocol in this layer.

Physical Layer Two examples are the IEEE 802.3 standards and the IEEE 802.11b standard which defines the physical layer of wired Ethernet and respectively wireless Ethernet. The TCP/IP stack does not define any specific protocol in this layer.

2.1.3 Connecting networks

Internet is a huge network connected with many different devices. Networks can communicate on different layers in the TCP/IP stack.

Repeater A repeater is a device that works in the physical layer. It receives a signal on one end, regenerates it and sends it out on all the other ends.

Switch The term switch can mean different things. A switch can work on different layers, *here* when discussing the switch it will mean a layer-2 switch.

The switch connects multiple networks on the data link layer. It contains a mapping between physical addresses and its interfaces which it uses when it forwards packets. Unlike the repeater the switch only forwards a packet on the interface where the receiver may be found, only when it does not know where the receiver is it will broadcast the packet on all interfaces except from where it came.

Router A router works on the network layer and has an IP address for each interface. It can decide based on the IP address where to forward a packet. When the router forwards a packet it changes the physical source address and the physical destination address in it.

Proxy server A proxy is an agent given the authority to act on someone else's behalf. A proxy server is connected to two networks with one external address and one internal address.

There are different kinds of proxies, application level proxies and circuit level proxies. Application level proxies (or application level gateways) are for example HTTP proxies or FTP proxies. Circuit level proxies support multiple applications.

Forwarding When packets are relayed from one network to another they are forwarded. A packet can be forwarded on different layers as described above. In the network layer there are different forwarding mechanisms such as unicasting, broadcasting, multicasting and anycasting.

Unicasting Forwarding packets from one host to another host.

Broadcasting Forwarding packets from one host to all hosts on a broadcast domain.

Multicasting Forwarding packets from one host to a group of interested hosts.

Anycasting Forwarding packets from one host to hosts defined by each destination address, but only one of these hosts is chosen to receive the packet at a time.

Port forwarding Port forwarding is also referred to as tunneling. With port forwarding a network port is forwarded from one network node to another. This technique makes it possible to allow remote computers connect to a specific host on a LAN.

With the help of tunneling IPv6 over IPv4 [11] a smoother transition into IPv6 is possible.

2.1.4 Name Server

A Name Server keeps a mapping between network dependant identifiers and human readable identifiers.

DNS The major naming system of today is the Domain Name System (DNS) which uses a hierarchical name space. The hierarchical name space, as opposed to a flat name space, gives a more structured way of assigning names. With a flat name space there are no connections between names. With a hierarchical name space there is a structured naming scheme to follow and the names are interconnected. For example, www.imit.kth.se and www.it.kth.se are two computers under the kth.se domain.

The objective is today merely to simplify the usage, a name is more easily remembered than an ip address but using the ip address gives the same result - www.it.kth.se or 130.237.203.50.

2.2 Definitions

In the context of this paper we will use our own definitions of mobility and mobility management.

2.2.1 Mobility

We define mobility as a move of any kind, anywhere, at any time. From our point of view mobility itself is not a problem, mobility will be accepted as a fact and not something we should try to avoid.

Examples of mobility, with our definition, can be the move of a laptop from one network to another, but also a move of a file from one folder to another.

2.2.2 Mobility Management

We define a mobility management system as a system responding on mobility of the kind described above giving adequate reactions on any event taking place in the system.

2.2.3 Endpoint

An endpoint is in our context a folder with a name according to our chosen naming scheme, described in section 3.3.1. The endpoint, as it is a folder, is possible to move from one computer to another. The endpoint can for example be stored on an usb-stick.

2.3 Motivation

There are many scenarios that would motivate the use of a mobility management scheme, below two different scenarios of different mobility types will be given and explained.

Disconnection tolerant mobility

> You come to work at eight o'clock in the morning and you sit down in your office. You always start the work day by booting up your laptop which by default always starts the applications you use, ftp and ssh clients. You have a meeting at nine o'clock but you figure out that you might have time to do a bit of work before going to the meeting in the building across the street.

At ten to nine you close the lid on your laptop to bring it with you to the meeting. In your building an old IPv4 network is still used while in the other building a new IPv6-only network has just been taken in use. When you cross the street your laptop looses connection to the wireless access point. Your laptop recognizes the lost connection and therefore suspend all ongoing sessions after saving the state of each session.

A few minutes later when you are in the other building on your way up to the conference room your laptop is given an IPv6 address. The session layer recognises the resumed networking connectivity and that the ip address is of a different type. The session layer notifies the name server of its new network address and resumes the connectivity of the ftp client and ssh client at the preserved state.

When at place in the conference room in the other building, you open the lid of your laptop. Both the ftp client and ssh clients are up and running from the state which they lost connection. You do not need to restart the applications or login - it is all done automatically.

The example scenario above is supposed to give you a hint on how the world of networking and mobility is progressing and even how the definition of mobility is changing as described in section 1, *"Our definition of mobility concerns the change of network address and also location while still keeping all connections. We also extend our definition of mobile objects to not only refer to computers but also to users, contents and applications."*.

Starting from the beginning of Internet, one could perhaps argue that the mobility solution of yesterday was in fact Internet itself. With web servers, ftp servers and e-mail the mobility problem with sending and spreading information fast was solved.

The mobility solution of today which is widely accepted and standardised is Mobile IP. The mobility solution of tomorrow is still work in progress and with many things to consider. There is a great need for mobility; when using your mobile phone today you do not expect it to disconnect when changing base station, why should not the same demands be put on computer networks?

To support seamless mobility certain requirements must be fulfilled. Banal requirements such as making sure computers supports the Internet Protocol stacks in use and more difficult requirements to realise such as how to overcome the built-in restrictions in sockets which prohibits rebinding. Other issues such as disconnection and reconnection need to be handled smoother to achieve a higher robustness and reliability of connections.

Reachability

You are hiking in a primeval forest watching rare birds, your friends playfully call you the ornithologist. For you, however, this is something you take very seriously. When walking around in the forest you see something in the corner of your eye flapping by. The bird you just caught a glimpse of, you need to get closer to see if your suspicion is right. Silently you move to the tree where the bird is resting at the moment. You take out your camera to zoom in and you get the shivers, this is what you suspected - an ivory-billed

> woodpecker[1], a very rare bird only seen on a very few occasions the last decades. You are thrilled with the sight of the bird and need to share this moment with your friends at the local amateur ornithologist association. You connect the camera's usb-cable to your mobile phone and the picture is now accessible from all browsers on the Internet and your friends can now confirm your discovery.

This scenario was supposed to give a view of mobile content. When typing in a website address in a browser it assumes that the website resides on a stationary computer, on a specific mounted harddrive, in a specific folder, in a specific file. But, as already said, content is not static anymore, harddrives are not static anymore and computers are not synonymous with desktops. A laptop can easily be moved from one network to another and will be given a new ip address. A harddrive can be unmounted on one machine and mounted on another machine with a different ip address. A file can be moved from one folder to another and even though this is just in one machine on one harddrive the linkage might still be broken and the content unaccessible. With a website address such as "http://www.host.com/manual/index.html" the only part of it that is constant is the name. "www.host.com" is linked to an ip address, "manual" is mounted on a harddrive and "index.html" is in a folder and they can all change location.

What is ingenious with the scenario is what the ornithologist actually is publishing, it is not the machine or a complete website - it is just the picture. We are lacking a word to describe what the picture is and therefore we define a new term, *cyber-object*. The picture is the cyber-object here, but why limit us to files. There is no reason to define the cyber-object of a special type, it could be just about anything limited to the smallest element in the computerized world - a bit. Even though a bit might be seen too far-fetched there are programs such as bittorrent[2] which divides files into smaller pieces.

2.4 Problem statement

Today when setting up a connection we choose which protocols to use and cannot change it while the connection is still ongoing. The Session Layer [5] rests on the belief in a protocol stack allowing rebinding in any level between protocols while still staying connected.

The session layer is a layer between the transport layer and application layer in the TCP/IP stack model. For applications to be supported by the session layer they need to initiate their connections through the session layer, thereby making the specific transport and network protocols transparent from an applications point of view. The session layer provides the features of suspending and resuming connections, if an object looses its network connection the state of all its sessions are saved and when it later regains its network connection all sessions are resumed from the preserved states.

These features will be implemented in the Session Layer Gateway (SLG). The SLG will forward connections between hosts on different networks but not necessarily with equal addressing schemes. When forwarding between different networks using different addressing schemes a translation mechanism is needed.

[1] http://www.birds.cornell.edu/ivory/
[2] http://www.bittorrent.org/protocol.html

Using the session layer all the way, from client through gateway to server, the ability of switching underlying transport and networks protocol will be made possible.

2.4.1 Requirements

The following functionality will be provided in the SLG:

- Forwarding mechanism in the Session Layer between:
    - IPv4 - IPv6 nets
    - IPv6 - IPv4 nets
    - private - (IPv6/IPv4) public nets
- Non-functional requirements put on the SLG are:
    - Should be able to handle 10^6 simultaneous sessions
    - Packet forwarding latency should be the same as for a router

2.5 Expected results

The expected results of this thesis will be a SLG supporting the requirements stated above with the expected results listed below:

- Survey on related technologies
- Understanding of the previous work - session layer design, implementation and objectives driving this work
- Design of the SLG through use cases
- Prototype implementation of the SLG
- Commented code and explanatory documents about the SLG
- This thesis

2.6 Evaluation methods - how to evaluate

The SLG will be evaluated with respect to functionality including error handling. It will be tested to take the appropriate action according to the state diagram (see section 3.5) and flow charts (see section 4.3).

2.7 Approaches to mobility management

In the research community there are many suggestions on how to solve this mobility issue. In these sections some of the approaches on different layers in the TCP/IP-stack will be described. In [31] the "mobility layer" is discussed and where it is best suited to be implemented. There are drawbacks and benefits with all solutions but what is pretty clear is that the higher in the stack mobility management is implemented the more functionality may be offered in the complete solution. In the discussion of related work (section 2.8) this is explained.

2.7.1 Mobile IP

Mobile IP [21] provides mobility at the network layer. Mobile IP introduces two new entities: Mobile Node and Home Agent.

Mobile Node The host that changes point of attachment from one network to another.

Home Agent Located on the home network to tunnel datagrams to the Mobile Node when the Mobile Node is in a different network. When a Mobile Node changes point of attachment it updates the Home Agent. The Home Agent itself is given a static IP address and is always reachable.

When a mobile node changes point of attachment the home agent will relay all packets to it. When a node on the home network sends a packet to the home address of the mobile node the home agent intercepts and forwards all the packets to the mobile node. When the mobile node wants to send packets to its home network it will change the source address of it to the address of its home agent. The node receiving these packets will not know that the mobile node currently resides in another network.

2.7.2 Host Identity Protocol (HIP)

HIP [26] provides mobility management in between the network and transport layer. HIP introduces a new name space where a host should no longer be identified by its IP address but by a Host Identifier (HI). Within the Host Identity Namespace every HI represents a globally unique name, by letting the HI be the public key of a public key pair. With HI comes the Host Identifier Tag (HIT) which is a 128-bit cryptographic SHA-1 hash over the HI.

With the new name space this also means that the transport layer will bind to the HIT. This gives the possibility of process migration, IP addresses can be changed by underlying protocols and HIP will send a re-address packet to inform its peer of the new address. The reason for sending this packet is a security measure to avoid DoS flooding attacks. HIP needs to check that a mobile node actually is reachable before sending data.

2.7.3 Migrate

Migrate [2][1] provides mobility management at the transport layer. Migrate is introduced as a new option in the TCP header. A TCP connection is identified by <source address, source port, destination address, destination port>. When setting up a new TCP connection the Migrate permitted option is set which permits the TCP connection to migrate at any point time. As part of the initiation of a TCP connection with the migrate permitted option a *token* will be negotiated. After negotiating the token a TCP connection can be either identified as above or by the <source address, source port, token>.

When migrating a TCP connection the mobile host will send a Migrate SYN packet containing the token identifying the previous connection to the fixed host. The fixed host will reply with an ACK/SYN packet acknowledging the last byte it received and implicitly acknowledging the SYN packet, if the Migrate SYN packet came from the right peer. The mobile host responds with an ACK and

the TCP connection is migrated and resumes from the last acknowledged byte in the TCP window.

By only allowing the migrate permitted option being sent in a SYN packet and by creating unguessable secure tokens the risk of TCP connections hijacking are minimised.

An optional session layer is introduced, which will give the application an abstraction of the lower layers. If the application is aware of the session layer it can join together related transport-layer connections to create a session ID. There is a session manager managing the events happening in lower layers such as change of address.

2.7.4 Reliable Network Connections

Reliable Network Connections [29] are two approaches to mobility at the transport layer. The two approaches both provides mobility in the user space.

Rocks Reliable sockets (rocks) is one of the approaches. Rocks provides a Rocks expanded API (RE-API) of the ordinary sockets API.

To establish a communication session through rocks a test for interoperability is done. If the corresponding peer does not support RE-API it will revert back to ordinary sockets. If it supports the RE-API, a data connection is established with TCP and an identifier for the connection is set between the peers. Finally, a control socket is set up using UDP for sending control messages.

Rocks uses the control socket to send heartbeats, periodical messages to check that a connection is still alive. If it is not alive rocks switches to a suspended state and tries to reconnect to the other peer.

Racks Reliable packets (racks) is the second approach. Racks is a packet filter between the local socket and the network layer and gives the same functionality as rocks. Instead of intercepting and modifying the packet through RE-API it intercepts and modifies the packet through a filter.

Instead of a user calling the RE-API for the initialisation procedure of a connection, racks will insert a message in the send stream. This message will appear to have come from the local socket. To preserve the consistency with the local socket racks needs to rewrite the source address, source port and sequence number for every packet to those expected by the receiving local socket.

2.7.5 Session Layer Mobility

Session Layer Mobility (SLM) [6][25], which is not to be mixed up with the Session Layer, provides mobility at the session layer. SLM does not treat the mobility between end hosts but the mobility of streams, an open data connection is a session and can be moved.

The SLM uses a User Location Server (ULS) to keep track of its mobile hosts, similar to the HA used in Mobile IP. To find the location of the Mobile Host a client can query an extended DNS which will reply with the location of the ULS. The location of the ULS can also be sent in the set-up phase between the session layers.

Mobility is hidden from applications and instead of using a direct communication link it uses proxies.

2.7.6 MOON

MObile Overlay Network (MOON) [12] provides mobility at the session layer.

The architectural model that MOON proposes is close to the one used in cellular networks. Two routing entities exists in MOON: the Enhanced Gateway Router (EGR) and the Enhanced Access Router (EAR). EGR is the border router of a domain and connects the domain to Internet. EAR is the router to where computers connect and is enhanced with location and authentication functionalities, this type of router functions within the domains. In the Mobile Host (MH) a session layer is implemented which manages the traffic. Along with the session layer also comes a session layer address space, handing out session layer handlers. These handlers encapsulates all the traffic.

To deal with applications only supporting the legacy TCP/IP stack a virtual interface is set up with a static IP address. All applications refer to this virtual internal IP address and the external IP address can change without the knowledge of the applications.

2.7.7 MSOCKS

Mobile SOCKS (MSOCKS) [9] is built upon SOCKS and provides mobility at the session layer.

MSOCKS uses proxies to provide mobility to nodes hiding the mobility from applications. MSOCKS uses a technique called TCP splice which allows an endpoint to believe that it is an end-to-end conversation even though there is a proxy in between. The TCP splice method forwards all packets received so there is no need to alter applications.

Between the application and the kernel on the mobile node sits the MSOCKS library. The MSOCKS library provides the applications with an interface identical to the Berkely Sockets API but it actually intercepts the connections and all data sent. This allows the proxy to change connection point and accordingly its ip address without the knowledge of the application. Upon reconnection the MSOCKS library handles it by unsplicing the connection and replacing the old outbound connection with the new one.

2.7.8 The Session Layer

Prior to this thesis an implementation of the Session Layer at endhosts were done [5]. The session layer provides mobility management in the session layer.

The session layer requires modifications in the TCP/IP-stack to support suspend and resume mechanisms for endpoints. Endpoints are no longer only computers, as mentioned earlier, but can also be users, content and applications. For an application to be able to take advantage of the session layer it must no longer bind to an ip address but to an endpoint name. This requires that applications are changed to be able to use the session layer.

If two endpoints have set up a communication session and one of the endpoints disconnects, the endpoint is shortly suspended and the state of the communication is saved at both sides. During connection time endpoints synchronise by sending checkpoint messages to check that data status is consistent on both sides and to allow data recovery if not. When the endpoint later reconnects the communication is resumed from where it stopped, the last checkpoint.

Implementation The session layer is implemented in the kernel of Linux 2.6.15 with a user space API defined.

Session Name Server With the session layer a new naming system is introduced, the Session Name Server (SNS). The SNS gives any object (endpoint) a name and allows it to update its ip address in real time.

When an endpoint register itself it first sends a message to the SNS. When it has moved it updates the SNS with its new ip address. When a client wants to reach an endpoint it sends a message to the SNS who replies with the ip address.

2.8 Discussion on related work

In this section we will be comparing the different solutions from a functionality point of view, not what is needed to be changed in order to make the solution work.

Each of the approaches to mobility management described above provides its own definition of mobility and accordingly its own specific solution. What we are looking for is a mobility management solution which offers transparency to applications and expands the concept of mobility to various objects.

The solution to mobility management at network layer which is standardised is Mobile IP. With Mobile IP it is not possible to keep connections when switching networks, this means that the state of the connection is lost.

In MSOCKS, SLM and Mobile IP proxies are used. It is always the proxy acting on the endhosts behalf and it introduces an unnecessary complexity of networking and also makes it less flexible. For SLM and MSOCKS it is possible to suspend and later resume a connection, a feature transparent for applications.

Racks and Rocks are implemented in user space and supports suspend and resume as well as the change of ip address.

Migrate's approach relies on one host being fixed and therefore does not offer mobility for two parts in a conversation.

The HIP approach is under standardisation by the IETF.

If comparing all of these with respect to functionality we find that mobility in our sense is not fully covered. Mobile IP does not support suspend and resume of applications. MSOCKS and MOON both rely on a proxy setup which makes the real underlying interface's ip address exchangable. HIP and SLM identify their devices by respectively a HI and an ULS. All of these solutions are using static names when setting up a connection which allows them to move their devices while the devices' sessions are ongoing.

A topic of debate is also whether an introduction of a mobility management solution should be complemented by applications, should they or should they not be aware of mobility. Applications specifically made to support mobility can provide features such as suspend and resume to users and not just be dealing with a worst case scenario of the cable being unplugged. We see the mobility awareness of applications as an important feature. For SLM, applications are not aware of mobility and can therefore not provide such features to users.

Migrate does so far seem to be the approach most similar to ours as it allows process migration of individual sessions. But Migrate is also a TCP

specific implementation and relies on one host being fixed. As stated in the introduction to this chapter, the higher in the stack that mobility management is implemented the more functionality may be offered in the complete solution.

2.9 Development environment

2.9.1 Hardware

As for hardware I am using three networked computers, one acting as a gateway with one computer on each side, but they need not necessarily all be physical computers. The hardware computers will be running User Mode Linux (UML) as it is faster to boot and edit. The UML machines will all be networked through Ethernet bridging. In figure 3 the setup is displayed.

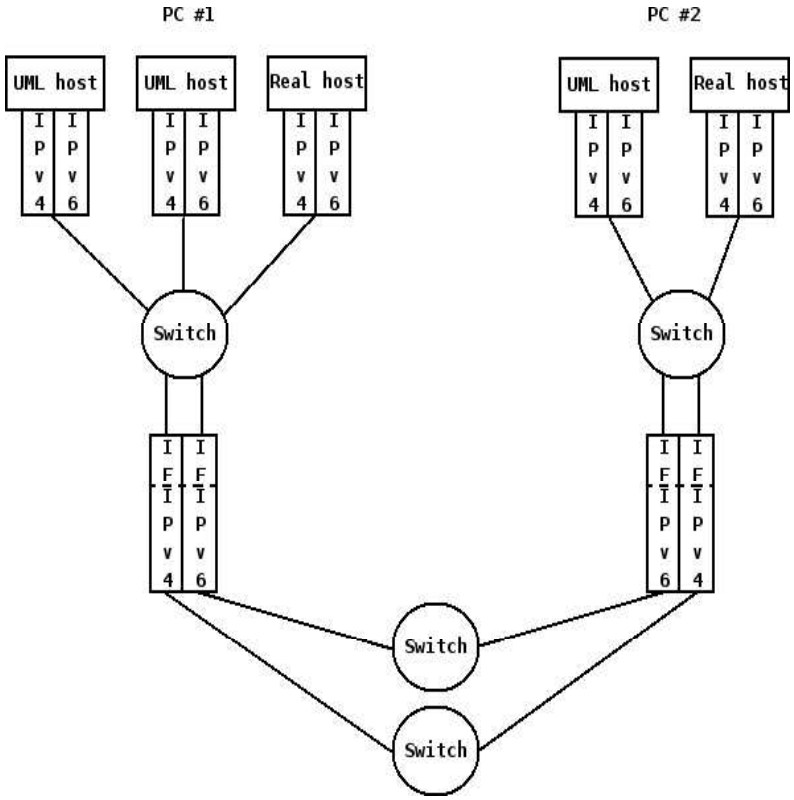

Figure 3: Development environment

2.9.2 Software

The computers run Gentoo Linux 2.6.15 and the compiler used is the GNU C Compiler.

As part of this work will be to forward connections from IPv4 to IPv6 networks and vice versa, a requirement is that all machines must run a dual TCP/IP stack.

2.10 Conclusions

In this section some important mobility approaches have been explained. Today, a vast number of approaches are proposed with their specific benefits and drawbacks. It is clearly possible to conclude that mobility is an important issue of the Internetworking of today.

With our approach mobility and mobility management takes a leap forward. Our approch expands the mobility concept and introduces new objects such as endpoints. Sessions between endpoints can be suspended and resumed, endpoints can be moved from one network to another. The TCP/IP stack is extended to support the session layer and applications need to be changed to bind to endpoint names instead of ip addresses to take advantage of the extended mobility features. Internet is old and has with regard to mobility outgrown itself a long time ago. With the fantastic and fast development of Internet and applications during the years it is now time to update it for the present and the future use of it.

3 Method - Design

This part of the thesis focuses on the design of the SLG. With design we mean the schematical view of the SLG, design choices made and rejected as well as reasoning.

The reason why two endpoints want to connect is that one of the endpoints is running a service that the other endpoint wants to use, for example a web server. From here on we will refer to the two endpoints as client and server.

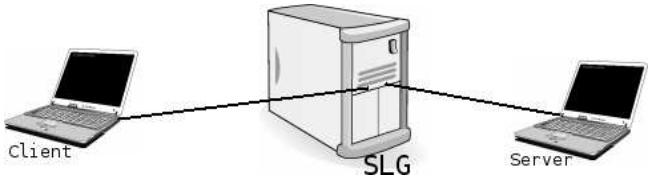

Figure 4: Example of Server-Client setup

3.1 Forwarding connections

The main idea of the SLG is to forward connections between endpoints on different networks. It is up to the endpoint to decide when to use the SLG to forward its connection. There are a couple of different cases to consider when an endpoint need to forward a connection and when it does not need to forward a connection. An overview of the different cases can be seen in figure 5. The SLG decides based on what kind of network the client and the server resides on which of the different protocols it should use to the respective recipient. Important to point out is that, as today, the applications will still see the communication between themselves as point to point.

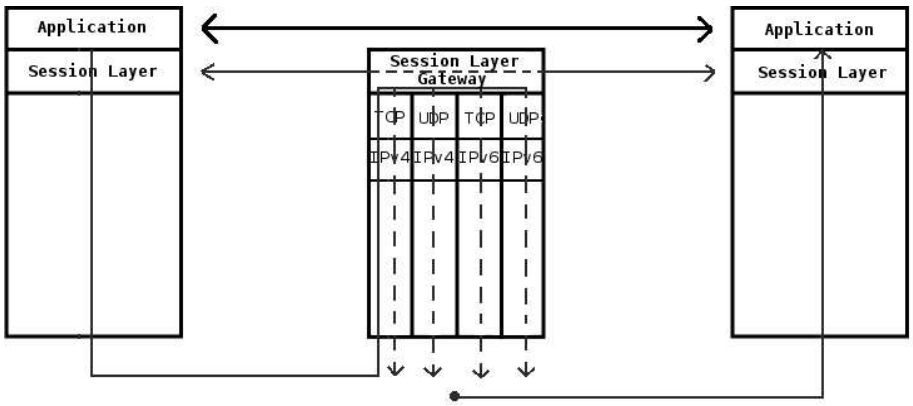

Figure 5: Schematical view of the communication process between two peers through the SLG

Table 1: Different cases of forwarding connections on networking level

Case	Host 1	↔	Host 2
#1	IPv4Pub	↔	IPv4Pub
#2	IPv4Pub	↔	IPv4Pri
#3	IPv4Pri	↔	IPv4Pri
#4	IPv6	↔	IPv6
#5	IPv6	↔	IPv4Pub
#6	IPv6	↔	IPv4Pri

3.1.1 Forwarding connections on network level

The different cases of forwarding that the SLG should be able to handle on networking level are listed in table 1. IPv4Pub refers to a public IPv4 address while IPv4Pri refers to a private IPv4 address. The private addresses are defined by IANA [32] as these three blocks of addresses: 10/8, 172.16/12 and 192.168/16.

Case #1 and #4, IPv4Pub - IPv4Pub and IPv6 - IPv6

These cases are trivial because forwarding through SLG is not necessary. With public IPv4 addresses and IPv6 addresses, which only exists as public, it is not possible to make a mistake on the identity of an endpoint. However, an endpoint may still wish to communicate through the SLG due to different reasons, a reason e.g. could be to hide its ip address from the other peer

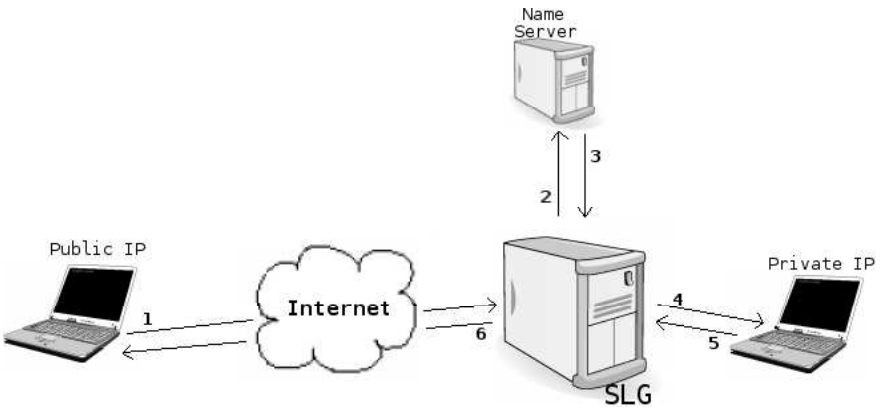

Figure 6: Public to Private network through SLG in setup mode

Case #2, IPv4Pub - IPv4Pri

With one endpoint residing on a private network no direct connection to this endpoint from another private network or a public network can be made. The standard solution in the legacy Internet would be to use Network Address Translation but this will instead be handled by the SLG. In figure 6 the step by step communication process when an endpoint with public ip address connects to an endpoint in a private network can be seen:

step 1 Public endpoint connects to the SLG

step 2 SLG asks the name server for address of private endpoint
step 3 Name server replies SLG
step 4 SLG connects to private endpoint
step 5 Private endpoint replies SLG
step 6 SLG replies public endpoint

Case #3, IPv4Pri - IPv4Pri
With two endpoints residing on two different private networks no direct connection between these two endpoints can be made. This will make both endpoints communicate through their respective SLG and make the SLGs communicate with each other. For the session layers at the endpoints this will be completely transparent.

Case #5, IPv6 - IPv4Pub
Communicating between two public addresses of different address schemes requires a translation mechanism. There should be no difference when mapping an IPv6 address to a public IPv4 address than between two public IPv4 addresses or two IPv6 addresses.

Case #6, IPv6 - IPv4Pri
The case when an endpoint with an IPv6 address tries to connect to an endpoint in a private network it is no different than when an endpoint with a public IPv4 address tries to connect.

3.1.2 Buffering

Forwarding packets between networks with different speed might cause an unbalance with respect to in and out ratio. This can be helped by using buffers, but as the session layer uses the TCP protocol there are already buffers in the transport layer. TCP also controls the amount of data allowed to be sent with its window size. If the window is full no more packets are to be sent.

3.1.3 Checkpointing

Checkpointing is made to check that data is consistent on both sides and to be able to recover data if data was lost during transmission, see section 3.4 in [5] and section 2.1.1.

The checkpointing is done between the endpoints and does not concern the SLG directly, though with the checkpointing somewhat expanded the SLG could be a part of it. The advantages would be an extra entity controlling the execution state of the transfer. The disadvantage would be that SLG would require more buffers.

3.1.4 Mapping client and server

The SLG needs to keep track of communicating clients and servers. From the session layer every session receives an unique session id. This session id is passed between client and server when communicating, letting the session layer know which session this packet is intended for.

When the SLG intercepts and re-packet packets it also need to know which session the packets are intended for. With the help of the session id the SLG maps clients to servers in a session list where entries based on the session id are unique.

Table 2: Example of server-client session mapping

Session ID	Server	Client
45810	server@example.com	client@example.com
21643	bob@example.com	alice@example.com

3.2 SLG mechanisms

The SLG's main objective is to forward traffic between endpoints. But, there are additional mechanisms that the SLG by definition provides and others that can be added.

3.2.1 Bridging heterogeneity

Bridging heterogeneity refers to bridging different types of address schemes. This is a default functionality in SLG, as described above with the different cases on networking level. This could also be extended to support bridging on the transport layer, see section 3.2.4.

3.2.2 Network Address Translation - "NAT"

NAT is widely used today to allow hosts on private networks to communicate with the outside networking world. When using the session layer the idea is that traffic from private networks by default will go through the SLG. If the SLG will handle all session forwarding between endpoints there is no need for an extra NAT service.

3.2.3 Privacy

Privacy is one built in mechanism that the SLG provides, by unpacking and re-packing the packet before forwarding it, see figure 7. Endpoints do not know its peer endpoint's ip address and is therefore only directly communicating with the SLG. This is also valid for endpoints with the same type of addressing schemes, even if it is certainly possible for them to communicate directly it will increase the privacy if they choose to communicate through the SLG instead.

3.2.4 Forwarding connections on transport layer

The different cases of network level forwarding are explained but in the SLG there is also room for using different transport layer protocols. Seeing that the SLG works on top of the transport layer it can use a different transport protocol on the client side than on the server side, as seen in figure 5. There are many different transport protocols and the two presented in that figure are only an example.

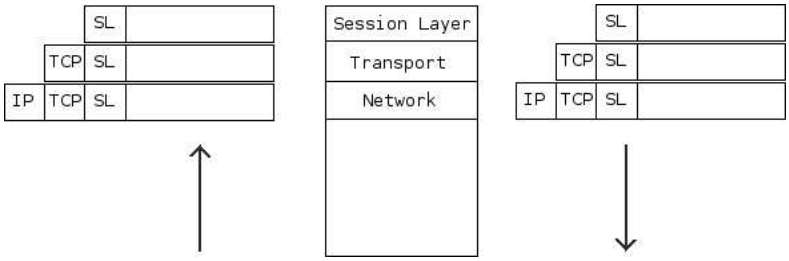

Figure 7: Unpacking and re-packing a packet

TCP [15], UDP [17], DCCP [10] and SCTP [27] are all different transport protocols with different advantages and disadvantages. DCCP and SCTP are relatively new protocols that probably will come in more use further on, DCCP was recently implemented in the linux kernel 2.6.14[3]. There is also the chance that new transport protocols might be developed and therefore the design of the SLG must be left open to be extended with support for additional transport protocols.

3.2.5 Caching

Caching is used to store data temporarily, in this case the session. The idea is to keep ongoing sessions between clients and servers through SLG separate. If a file is being transferred from server to client and the client of some reason suspends it does not necessarily need to stop the server to SLG communication. The server could keep sending the file and the SLG would store the data in a file on its harddrive. When the client in a later stage decides to resume its connection the SLG will send the rest of the file to the client without the server being involved.

3.2.6 Firewall

Firewall support is an extra mechanism that can be built in. Making SLG the gateway to Internet will not only be something possible to provide but something that will be taken for granted. Malicious users, viruses, masks and other threats should be blocked already by the SLG.

3.2.7 AAA

Authentication, Authorization and Accounting (AAA) [7] are additional services the SLG could run to enhance the access policy restrictions. The authentication part checks to see that an user is allowed to access the services requested. The authorization part checks to see which services an user is eligible for and acts upon that. The accounting part keeps track on the network resources used by an user.

[3] http://lwn.net/Articles/149756/

3.2.8 Endpoint resumption on different network address scheme

If two endpoints on homogeneous networks starts a session they do not need a SLG in between. If later one of the endpoints suspend its communication and moves to a different network there is now need for a SLG. In that case, the endpoint faces two problems. The first problem is how to find the SLG (see 3.2.9) and the second problem is how to resume a session through the SLG that never existed there.

The other way round is supposedly an easier task. The use of the SLG is transparent for endpoints which should make the move from communication through the SLG to direct communication possible straight off.

3.2.9 SLG discovery

When an endpoint discovers that the peer it would like to connect to resides on a network with different address scheme it needs to communicate through the SLG. The problem is how to find the SLG and there are different approaches.

static IP Statically assigning the IP for the SLG is not an approach fitting the session layer concept. Internet is, in our view, supposed to be dynamic in all ways possible and therefore static assigning is never good.

static endpoint name This is the equivalence of a static IP in the session layer world and is not fitting either, the same rule apply here as for static IP.

DNS SRV DNS SRV [4] is a DNS Resource Record for specifying the location of services. A client will ask for the service for a domain and will receive a reply with the name of the available server.

3.2.10 Moving Sessions

The SLG is not mobile itself, its position in the network is static and it is considered to be a part of the network infrastructure. By being a part of the infrastructure it ensures that both endpoints in a communication session can be mobile at the same time.

If endpoints move from one SLG to another there need to a be a way for the SLGs to communicate session states between themselves. However, these session states need to be sent in a secure way to authenticated as well as authorized SLGs to ensure that sessions are not hijacked by malicious users.

There are other challenges involved in the scenario of moving sessions between SLGs. The SLG needs to be told that an endpoint has moved to another SLG, otherwise it assumes that the endpoint after a time will return and resume its connections. If an endpoint move to another SLG it needs to retrieve the session states from the old SLG. This might incur some problems on how to get the previous SLG's name - the idea of the SLG is that it is supposed to be imperceptible to the session layers at the respective endpoints.

3.3 Naming

The naming convention as well as the name resolution decides what is meant by an endpoint name. To allow the level of mobility we set as a goal we require a good naming convention with an equally good name resolution

3.3.1 Endpoint naming

The design of the naming system is not straightforward because of the complex network structure that the endpoints entail. There are different types of endpoints: users, applications, services, computers and perhaps even more. Should all endpoints use the same type of naming convention - the question is whether one endpoint need to know what type of other endpoint it is communicating with.

There are some requirements that should be put on the naming convention. Names need to be unique, names should be easy to remember and easy to construct. An endpoint name should not easily be mistaken for being something else.

endpoint@domain

To name endpoints such as bob@example.com is one solution but the meaning of it is ambiguous. In this context bob@example.com is the name of an endpoint but in other contexts it could be an e-mail address or the syntax when bob tries to copy, download or remotely login with scp, ftp or ssh respectively.

3.3.2 Port number retrieval of services

There are different solutions on how to retrieve the port number of an endpoint. An endpoint is identified by its endpoint name for an application programmer but for the underlying structure an endpoint is identified by the combination of an ip address together with a port number.

Keeping port number at Name Server

The original port number communication is done with the Name Server. When a service is setup at an endpoint, it will inform the Name Server of its port number. When a client wants to connect to the service it will first retrieve the ip address of the server and then the port number of the service at the server. With both ip address and port number, the client can start a communication session directly with the service at the server.

Listen on pre-determined/standard port numbers

A different approach on how to communicate port numbers would be to listen on pre-determined ports or standard port numbers. If an endpoint runs a webserver, the standard port number is port 80. There would be no need to communicate this to the client. However, if the server would be running an application without a standard port number it would still somehow need to tell the server which port the service is running on.

Use the control port of the Session Layer
The third approach would be to use the pre-defined control port of the session layer to setup the session. A client that wants to use a service at a server would first connect to the control port to ask for the port number. The server would send a reply with the port number where the service is running. The client would connect to the server on the received port number.

The first and third approach will always work and will always work in the same way, no matter if the service is a web server on a standard port number or a user written application on a non-standard port number. The advantage with the third approach compared to the first approach is that the communication with the name server is halved, which leads us to the conclusion that the third approach will be the one to use.

3.3.3 Port number retrieval of control port

There are two different solutions how to get in contact with the control channel of an endpoint. Either a predetermined control port number can be used or an ICMP like procedure might be used. The different types of session packets using the control channel are described in more detail in section 3.6.1.

Predetermined control port number When one endpoint connects to any other endpoint the procedure is straight forward, it will always connect to the other endpoint on the predetermined port number. But to use a predetermined control port number we need to be certain that the number is not used by any other application, we would require an ICANN[4] standard.

ICMP The ICMP [18] server does not listen on any specific port which is a desirable ability. If the control channel was using the ICMP server there would be no need for a predetermined port number and there would never be an issue that the port might be taken. An endpoint would send the message to the endpoint without defining any port number and would receive a reply in the same way, an echo request/echo reply procedure.

Other benefits are that if the session layer was listening on a specific port number this might be transport protocol dependent. There are many different transport protocols and the session layer should be free to use whichever it finds the best in every situation.

In the prolongation the ICMP protocol might even be extended with headers supporting connect, suspend and resume.

3.3.4 Name Server awareness of SLG

With respect to name resolution the SLG is as any other endpoint and the name server will not be able to tell whether it is a regular endpoint or a SLG communicating with it.

If an endpoint decides to communicate through the SLG, the endpoint's ip address will be unknown by everone except the SLG. When a peer wants to communicate with the endpoint it will receive the ip address of the SLG to start the communication.

[4] http://www.icann.org

3.4 Gateway extension of the Session Management Protocol

The SLG and session layers on endpoints need to communicate at different stages in the communication process. Even though the SLG is supposed to be transparent to session layers at endpoints, the SLG itself needs to have defined what it should do in the different stages, that is how to respond to different types of messages. Communication between SLGs is also something that will be necessary, for example moving session states.

Setup When a client realises that the server it wants to connect to is on a different addressing scheme it needs to set up the communication process through the SLG.

Synchronisation When an endpoint moves from one network to another the state is saved by both endpoints. When the endpoint connect to a network and the connection resumes, a synchronisation message between the session layer on the endpoint and the SLG is sent. The synchronisation message informs the SLG the state of every connection.

Moving If endpoints have disconnected from one SLG, the SLG keeps the state of the sessions. If the endpoints later re-connect with the same SLG then it can just resume from where the session was suspended. However, if an endpoint resumes its network connection at another SLG that SLG needs to know the resume parameters. Therefore a mechanism for retrieving information from the old SLG is needed.

3.5 SLG states

The states in the state diagram (figure 8) are defined in the SLG as in the endpoints. The transitions in the state diagram are defined and explained below.

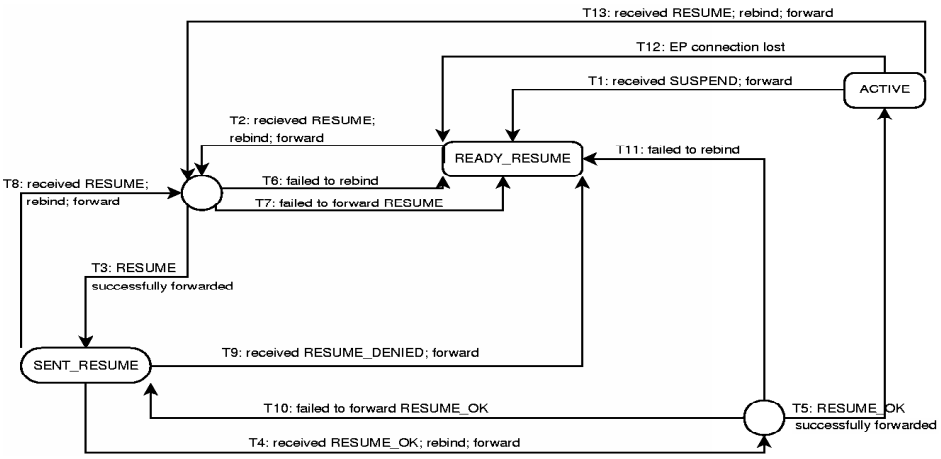

Figure 8: SLG state diagram

Transition 1

State ACTIVE
Event One of the communicating endpoints wants to suspend the session
Action SLG receives SUSPEND message and forwards it. Enters the READY_RESUME state
Description An endpoint wants to suspend and sends a SUSPEND message to the SLG. SLG passes the message on and goes into the READY_RESUME state. That is, just after forwarding the message the SLG is ready to RESUME the session.

Transition 2

State READY_RESUME
Event Received RESUME message from one of the endpoints.
Action SLG receives the RESUME message, it rebinds to the (possibly) new address and forwards the packet. Enters intermediate state RECEIVED_RESUME.
Description The RESUME procedure is commenced.

Transition 3

State RECEIVED_RESUME
Event Successfully forwarded RESUME
Action Enters SENT_RESUME state.
Description The message was successfully forwarded which also means that the SLG successfully bound to the (possibly) new address of the resume-initiating endpoint. Endpoint should now be informed that its peer wants to resume the session. SLG will wait for a RESUME_OK in state SENT_RESUME.

Transition 4

State SENT_RESUME
Event Received RESUME_OK message from endpoint.
Action Enters the intermediate state RECEIVED_RESUME_OK. SLG receives the RESUME_OK message, rebinds to the (possibly) new address and forwards the packet.
Description The RESUME procedure is almost completed, remains to successfully rebind and forward the RESUME_OK message.

Transition 5

State RECEIVED_RESUME_OK
Event Successfully forwarded RESUME_OK
Action Enters ACTIVE state
Description The RESUME procedure is completed, endpoints are informed and the session is now resumed. SLG will enter state ACTIVE and be ready for regular transfer of data.

Transition 6

State RECEIVED_RESUME
Event Failed to rebind
Action Enters the READY_RESUME state

Description If SLG could not rebind this is a network problem. SLG will enter READY_RESUME state for endpoints to try again.

Transition 7

State RECEIVED_RESUME
Event Failed to forward RESUME message.
Action SLG replies with a RESUME_DENIED message. Enters the READY_RESUME state.
Description SLG failed to forward the packet because of a network problem. It will inform the first endpoint that there is a communication problem with the other endpoint by sending a RESUME_DENIED message back. SLG enters READY_RESUME state and the endpoints can try again.

Transition 8

State SENT_RESUME
Event Received RESUME message
Action SLG receives RESUME message, will rebind and forward RESUME message. Enters RECEIVED_RESUME intermediate state.
Description SLG receives RESUME message in state SENT_RESUME, endpoints do not agree on who should resume the session. SLG does not want to interfere and forwards the packet.

Transition 9

State SENT_RESUME
Event Received RESUME_DENIED message
Action Forward RESUME_DENIED on and enter READY_RESUME state.
Description Endpoint will not allow session to be resumed and replies with RESUME_DENIED. SLG is made aware of this and enters the READY_RESUME state.

Transition 10

State RECEIVED_RESUME_OK
Event Failed to forward RESUME_OK
Action Enters the SENT_RESUME state
Description SLG could not forward the RESUME_OK message and enters the SENT_RESUME state waiting for the endpoint to try again.

Transition 11

State RECEIVED_RESUME_OK
Event Failed to rebind
Action Enter READY_RESUME
Description If SLG could not rebind this is a network problem. SLG will enter READY_RESUME state for endpoints to try again.

Transition 12

State ACTIVE
Event Lost connection with Endpoint
Action Enter READY_RESUME state

Description Endpoint disappeared without informing SLG. After TCP timeout SLG will enter READY_RESUME state.

Transition 13

State ACTIVE
Event Received RESUME message
Action SLG will rebind and forward the RESUME message
Description If SLG receives a RESUME message in this state, it should be treated as if the SLG was in READY_RESUME state. Probably the endpoint wants to inform of a new address and the session will not be able to RESUME until the whole RESUME procedure is completed.

3.6 Session Layer Packets

There are thirteen different types of session layer packets. All packets contain the session header and a possible payload. The session layer is communicating on two channels, a control channel and a data channel. The control channel is used for control messages and the data channel for data transfer. Some of the session layer packets are sent on the control channel and some are sent directly on the data channel.

3.6.1 Control channel packets

The packets sent on the control channel are packets that will either setup a new communication, suspend a communication that is already in action or resume a previously suspended communication session.

The SLG will always need the endpoint names of the receiving endpoint and of the sending endpoint. If the communication session was only peer to peer, the peer endpoint name would not be needed, but to support the SLG also the peer endpoint name is added in the payload of a control packet.

SESSION_CONNECT This packet is used to initiate a session. In addition to figure 9 it will contain the name of the service requested.

SESSION_CONNECT_OK This packet will confirm the startup of a new session. In addition to figure 9 it will contain the port number on which the service is running.

SESSION_CONNECT_DENIED This packet is sent if there is no such service running on the server or if the server does not wish to communicate with the connecting client.

SESSION_SUSPEND This packet will ask for suspension of a session by interrupting the sending and receiving sockets.

SESSION_SUSPEND_OK This packet will approve the suspension of a session.

SESSION_SUSPEND_DENIED This packet will deny the suspension of a session.

SESSION_RESUME This packet will initiate a resume of a previously suspended session. In addition to figure 9 it will contain the port number which the socket previously was using for the communication session.

SESSION_RESUME_OK This packet will confirm the resume of a previously suspended session. In addition to figure 9 it will contain a new port number which the service is ready to accept connections on.

SESSION_RESUME_DENIED This packet is sent if there is no such service suspended. Except from the obvious reason that there never was such a service this is handled differently in session layers on endpoints than in the SLG. If the SLG sends this packet the reason could be that it was in suspended state too long and the session timed out, it was assumed to have been moved to a different SLG. Sessions in the session layer on an endpoint do not time out.

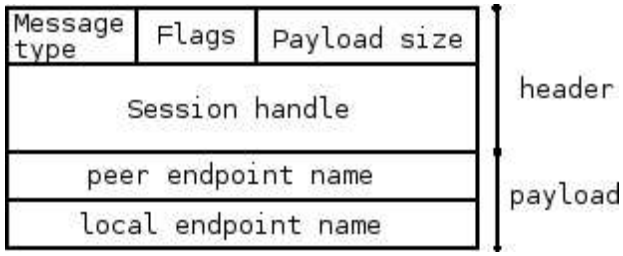

Figure 9: Control Packet with header and payload

When communicating through the SLG, the SLG will alter the contents in some of the packets. In the setup procedure it is vital that the SLG replaces the local port number of a server with its own port number, see figures 15 and 16 and the explanatory texts.

3.6.2 Data channel packets

The packets that are sent on the data channel are mainly transfering data and making sure that the data is consistent with the help of checkpoint packets. The packet that ends a communication session is also sent on the data channel.

SESSION_HEADER This packet will complete the session startup. SESSION_CONNECT and SESSION_CONNECT_OK has already been handled on the control socket and then a SESSION_HEADER will be sent by the client on the data socket. It only contains the session header and confirms for the service that the client is ready to receive.

SESSION_DATA This packet is the most common packet. A packet of this type contains the session header and data payload.

SESSION_CHECKPOINT This packet checks data consistency between client and server. A SESSION_CHECKPOINT packet can be sent either by the client or the server and is confirmed by a another SESSION_CHECKPOINT packet sent back. The packet contains the session header and a payload with the current position of the data stream on this specific endpoint.

SESSION_CLOSE This packet closes a session. When client or server is ready to close the session, a SESSION_CLOSE packet is sent. The packet is confirmed by sending a SESSION_CLOSE packet back. The packet only contains the session header.

4 Implementation

In this part of the thesis the implementation of the SLG is described as well as the alterations to the session layer at endpoints [5].

4.1 Scope

The scope of the project was to design and implement a SLG:

- The SLG should handle connections on both IPv4 and IPv6.
- The name resolution should be integrated with the session layer.
- The session layer on endpoints only supported IPv4 and had to be extended for IPv6 support.

Due to time constraints the SLG was implemented with basic functionality, as stated above. Other mechanisms described in section 3.2 are left for future work.

4.2 Implementation decisions

4.2.1 Alterations to the session layer at endpoints

When introducing the SLG in the session layer system some changes had to be made in the original session layer at endpoints to enable SLG support. Below the changes made are listed and in following sections more elaborate explanations are given.

Control packet payload Local endpoint name and peer endpoint name are always sent in the payload, see figure 9.

Service port number retrieval The service port number is sent back in the SESSION_RESUME_OK packet.

IPv6 support added

Name server itself Name server only saves the endpoint name, no service port number.

Name server communication Name server communication is done through user space.

These alterations have not changed the usage of the session layers which still, of course, function peer to peer.

4.2.2 Adding IPv6

The implementation of IPv6 in Linux has been based on IPv4 and therefore the use of IPv4 sockets and IPv6 sockets are similar [24]. However, IPv6 addresses are four times the size of an IPv4 addresses and therefore all data structures used need to be large enough to hold an IPv6 address. This was accomodated in Linux by adding extra data structures that supported IPv6. For IPv4 addresses we have *struct sockaddr_in* and *struct sockaddr*. For IPv6 addresses we have *struct sockaddr_in6* and *struct sockaddr_storage*.

The *struct sockaddr* and *struct sockaddr_storage* are generic data structures and must be used when calling bind[5] to bind a network address. In figure 10 and 11 the host type field (family) is a field that exist in all data structures. The host type is mapped from the real data structure into the castee data structure and is always accessible. By checking the host type the SLG can tell whether this is an IPv4 or an IPv6 address and therefore how to act.

One of the parameters passed to bind is the size of the data structure and as long as the passed size is not too small it will work. Therefore the passed size is always the size of the *struct sockaddr_storage*, because *struct sockaddr* will fit into it but not the other way round.

```
struct sockaddr{
  sa_family_t   sa_family;
  char          sa_data[14];
}
```

```
struct sockaddr_storage{
  short          ss_family;
  char           __ss_pad1[_SS_PAD1SIZE];
  __int64        __ss_align;
  char           __ss_pad2[_SS_PAD2SIZE];
}
```

Figure 10: Comparing IPv4 and IPv6 generic data structures

```
struct sockaddr_in{
  sa_family_t     sin_family;
  u_int16_t       sin_port;
  struct in_addr  sin_addr;
}
```

```
struct sockaddr_in6{
  u_int16_t        sin6_family;
  u_int16_t        sin6_port;
  u_int32_t        sin6_flowinfo;
  struct in6_addr  sin6_addr;
  u_int32_t        sin6_scope_id;
}
```

Figure 11: Comparing IPv4 and IPv6 data structures

4.2.3 Control channel threads - SLG daemons

The control channel are running on two threads, one for IPv4 and one for IPv6. They are both listening on the predetermined control port number, this is achieved by setting two parameters in the socket structure. The first is a parameter which allows a socket to bind an already bound port and the second parameter IPV6_V6ONLY sets the IPv6 socket to only react on IPv6 connections.

There is a possibility to let an IPv6 socket also respond on IPv4 connections by letting an IPv4 address be represented as an IPv6 address, that is the IPv4-mapped IPv6 addresses. An IPv4-mapped IPv6 address are written as $::FFFF:<IPv4-address>$. We have made a deliberate choice to keep IPv4 and IPv6 separate to avoid any conflicts and not incur any extra complexity.

4.2.4 Data channel threads

Similar to the control channel the data channel listens on two threads but whether they listen on IPv4 or IPv6 or not is decided when setting up the

[5]http://linuxreviews.org/man/bind/#toc1

data sockets. The reason that SLG listens on two threads is that communication between client and server is bi-directional, one thread listens on the server and one thread listens on the client. Both threads use the server and the client sockets, the server thread receives on the client socket and sends on the server socket while the client thread receives on the server socket and sends on the client socket, see figure 12.

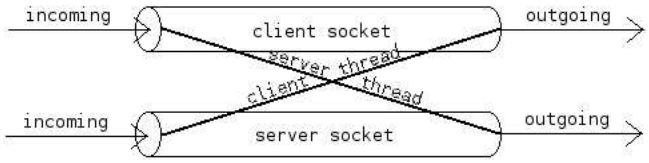

Figure 12: Data socket usage

4.2.5 SLG functions

All of the SLG functions are implemented in the kernel and there is no need for a user space API. The functions below are the three main functions that keep track on the traffic going in and out of the SLG.

session_forward_setup This function is called from one of the SLG daemons (IPv4 daemon or IPv6 daemon) when receiving a SESSION_CONNECT packet. Though, this function itself calls the function *session_pass_on* and when *session_pass_on* has terminated it returns here. Given that it returned successfully it will continue the setup procedure by passing on the received SESSION_CONNECT_OK message to the client. If everything goes well it will spoon a thread that will keep track and act upon the client-to-server traffic while this thread will keep track and act upon server-to-client traffic.

session_pass_on This function will receive the SESSION_CONNECT packet and set up a control socket to the server and pass on the packet. It will wait for a SESSION_CONNECT_OK packet to come back and if so set up the cs_conn structure (see 4.2.7).

forward_to_server This function will be listening on the client thread and act upon received messages.

4.2.6 Socket approach

For every call to the control channel two new sockets are started, the incoming socket and the outgoing socket. New sockets are always created to ensure complete mobility of endpoints since it is possible that an endpoint has changed ip address during control message exchange.

4.2.7 SLG data structure

There is need for a data structure to keep track of all important data that belongs to a thread, it is the *struct cs_conn* and consists the following variables.

struct endpoint *client_ep The endpoint that initiates a session.

struct endpoint *server_ep The endpoint that hosts a service which client_ep connects to.

struct session *session The session.

unsigned int server_port The remote port which SLG connects to on the server_ep.

unsigned int client_port The local port which the client_ep connects to on the SLG.

struct socket *out_socket The socket between the SLG and the server_ep.

struct socket *in_socket The socket between the client_ep and the SLG.

struct socket *tmp_socket When suspending/resuming we need to hold a temporary socket.

unsigned int session_close_client If true, SLG has closed the session with the client_ep.

unsigned int session_close_server If true, SLG has closed the session with the server_ep.

unsigned int session_close_client_recvd If true, SLG received a SESSION_CLOSE message from the client_ep.

unsigned int session_close_server_recvd If true, SLG received a SESSION_CLOSE message from the server_ep.

unsigned int resume_init_client If true, client is the initiator of the resume procedure.

unsigned int resume_init_server If true, server is the initiator of the resume procedure.

struct sockaddr_storage *out_daddr Belongs to the out_socket (above). The corresponding sockaddr_storage structure that belongs to in_socket is kept in the session struct.

4.3 Flow charts

The flow charts, figure 13 and figure 14, gives an overview how the session layer at an endpoint and at the gateway will react on a given event. In the following section 4.4 the different procedures will be explained in more detail.

When the session layer in the endpoint is receiving or sending data it is considered to be in the normal state. If an event occurs the different procedures will take place. The same is valid for the gateway.

Between a SUSPEND packet and a RESUME packet, an endpoint might have switched point of attachment. In the case of a cable unplugged in figure 13 or an endpoint connection lost in figure 14 the session layer will be waiting until the cable is plugged in or the endpoint is alive again. The only event where an ongoing session is closed is when a CLOSE packet is received, in the setup phase when a CONNECT_DENIED packet is received the session is never started.

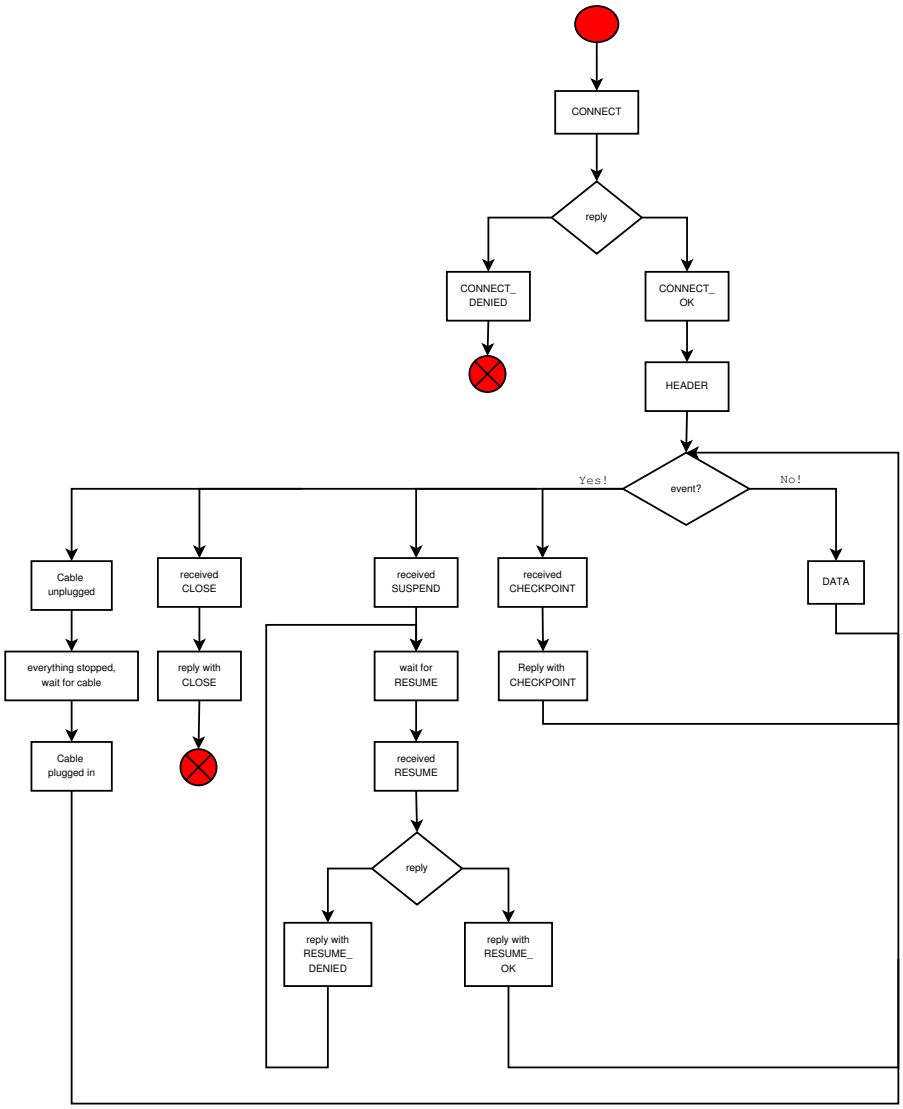

Figure 13: Endpoint Session Layer flow chart

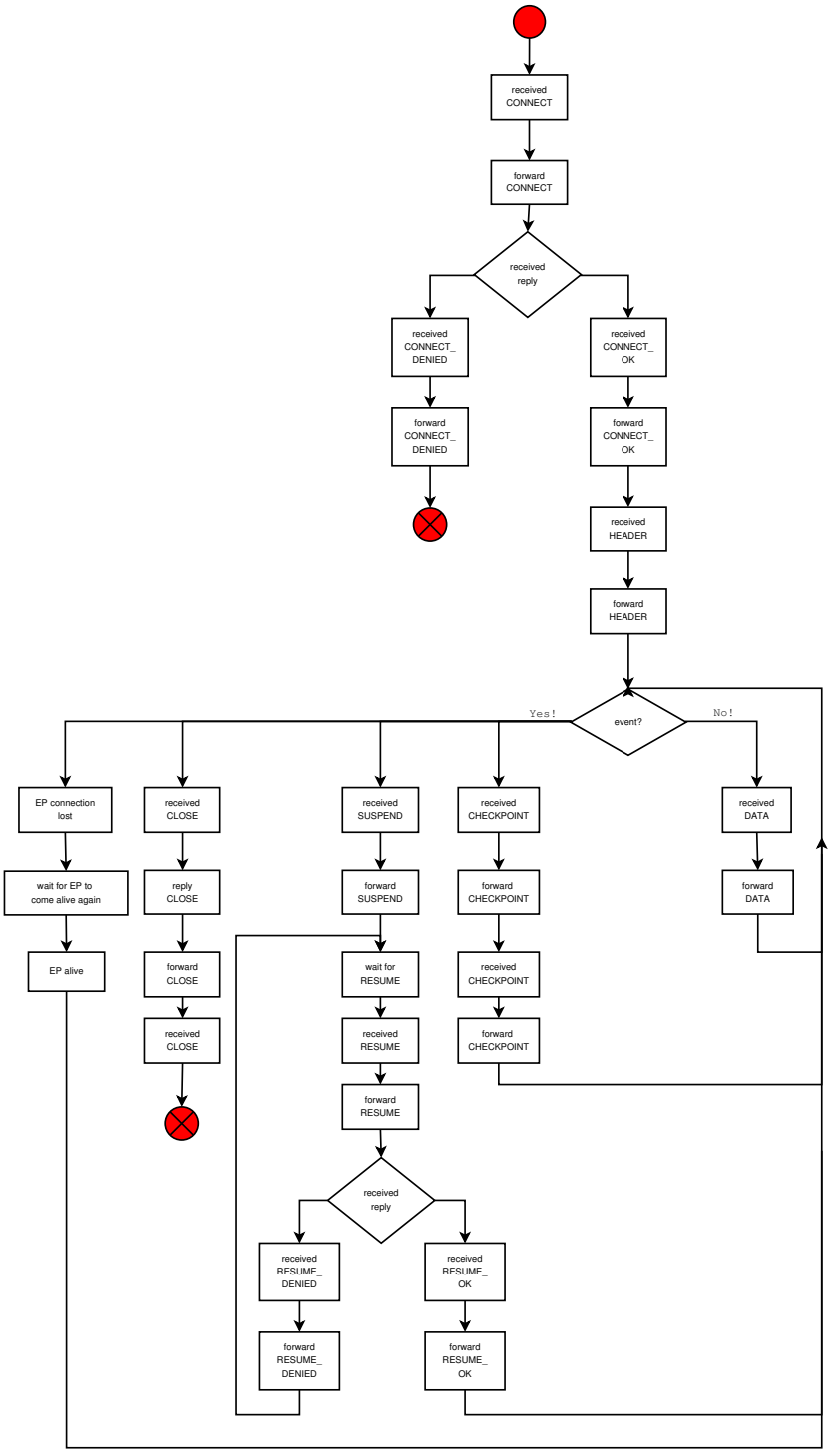

Figure 14: SLG flow chart

4.4 Phases of communication

There is one main rule that apply to the session layer, it states that a SLG must be transparent for an endpoint's session layer. Regardless of whether it communicates with its peer through the SLG or not, all types of communication is done exactly the same. Applying this rule for the SLG itself gives that the SLG should handle communication as smoothly as if it was not there. While it is important to point that out, it is still important to see what level of complexity the SLG brings which can be seen when comparing the figures below.

4.4.1 Setup procedure

The setup procedure is started when a client wants to connect to a service run by a server.

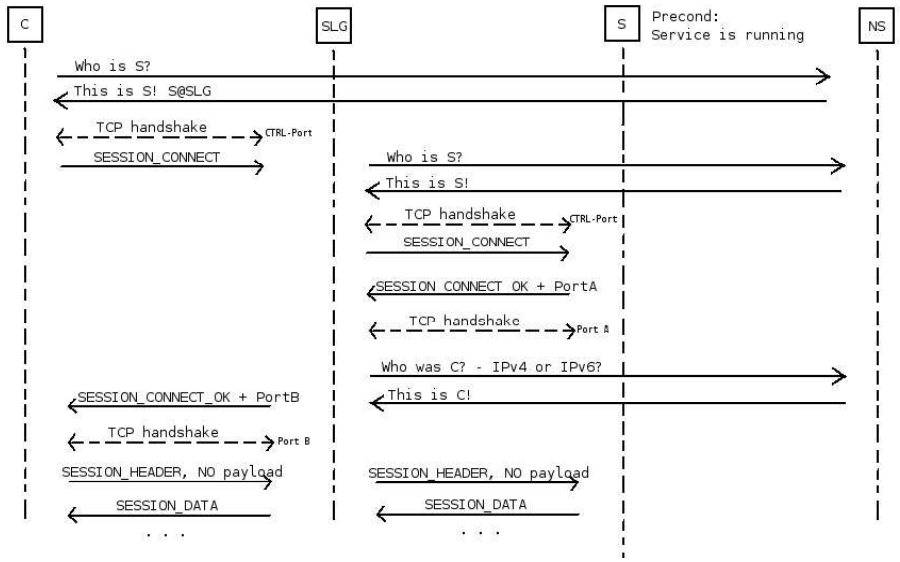

Figure 15: Setup procedure between Client and Server through SLG

with SLG The setup procedure between client (C) and server (S) through SLG is displayed in figure 15.

It is started when a C wants to use a service run by S, it needs to establish a communication session. First C will ask the Name Server (NS) for the ip address of S. If the ip address of S is of a different type than C uses itself it will involve the SLG in the setup procedure. After receiving the ip address of S it will establish a connection with the control port of SLG and send a SESSION_CONNECT message.

SLG will ask NS of the ip address of S and establish a new connection to the control port of S and pass on the SESSION_CONNECT message. If S has the service running it will reply with a SESSION_CONNECT_OK message containing the port number that the service is running on. SLG will receive the packet and start up a listening port of its own. SLG will then replace the port

number in the packet with its own. Then SLG will ask NS of the ip address of C and pass the packet on.

C will receive the packet and establish a new connection on the port and send a SESSION_HEADER message as a confirmation of the newly established connection. SLG will receive the packet and pass it on to S. S receives the packet and will directly start sending SESSION_DATA packets to C through SLG.

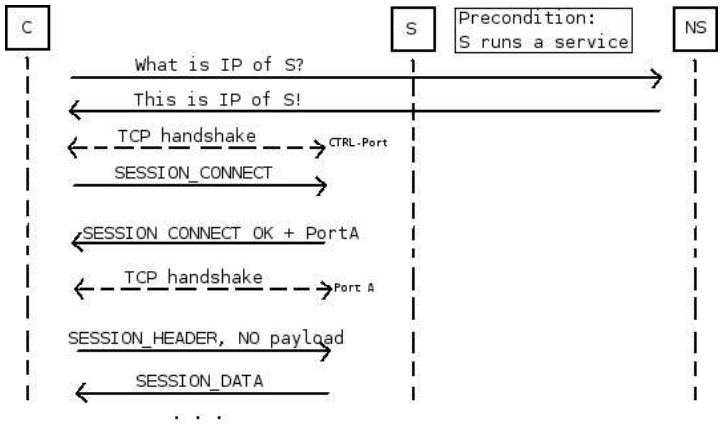

Figure 16: Setup procedure between Client and Server

P2P The setup procedure without the SLG is displayed in figure 16.

It is started when a client (C) wants to use a service run by server (S), it needs to establish a communication session. First C will ask the Name Server (NS) for the ip address of S. After receiving the ip address of S it will establish a connection with the control port of S and send a SESSION_CONNECT message. S will, if it has the service running, reply with a SESSION_CONNECT_OK message containing the port number that the service is running on. C will establish a new connection to S on the received port number and send a SESSION_HEADER message. The SESSION_HEADER message will confirm the new communication session and S will directly start sending SESSION_DATA packets.

4.4.2 Suspend/Resume procedure

When either a client or a server in an ongoing session wants to suspend the session it sends a SESSION_SUSPEND message. Later when the client or server wants to resume the suspended session it sends a SESSION_RESUME message. The figures explains what would happen if the server initiates a suspend message, but it would have been the same scenario if the client had initiated it instead.

with SLG In figure 17 the suspend and resume procedure with SLG is displayed.

It is started when the user application of some reason decides to suspend a session by sending a SESSION_SUSPEND message. This message is sent on

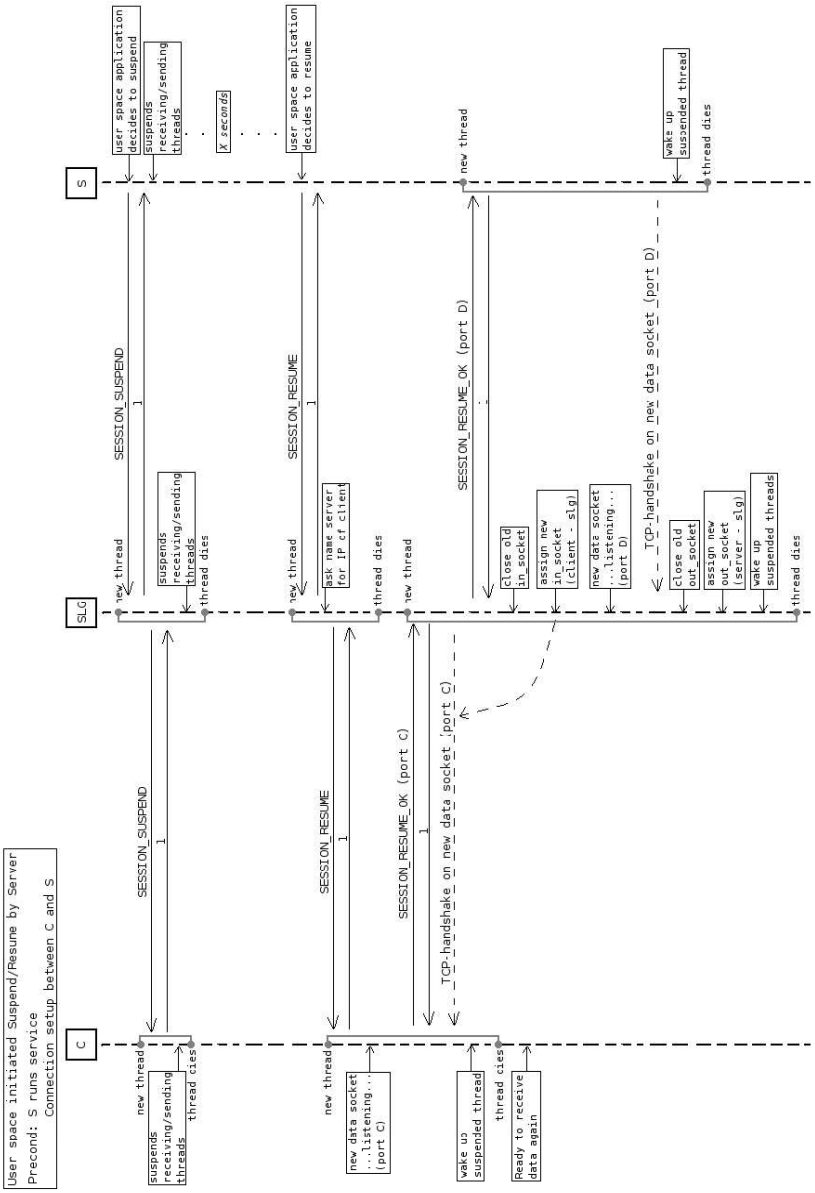

Figure 17: Suspend and resume procedure in SLG

the control channel and an acknowledgement (ACK) packet is sent back as a confirmation that the packet was received and that the session exists. When the server receives the ACK packet it suspends its threads. The SLG passes the received suspend packet on to the client. The client sends an ACK packet back and suspends its threads. The SLG receives the ACK packet and suspends its threads.

During the time of suspension it is possible for endpoints to move, to change point of attachment. When they later register themselves at another location the name server will be informed. Therefore the SLG will always ask the name server for the ip address of the endpoint that it will forward packets to when a session is resumed.

An amount of time has passed and the server decides to resume the session, therefore it sends a SESSION_RESUME message. The SLG receives the message and sends an ACK packet back. The server is now waiting for a SESSION_RESUME_OK message. The SLG passes the packet on to the client, after retrieving the possibly new ip address of the endpoint, and receives an ACK packet. The client has received the packet and sets up a new data socket on a new port C. It sends a SESSION_RESUME_OK packet back with the new data socket port number attached and receives an ACK packet. SLG connects to the new data socket on the client and passes on the SESSION_RESUME_OK packet after changing the attached port number. SLG receives an ACK packet from the server and the server connects to the new data socket on the SLG. After connection the threads are awoken and the session is resumed.

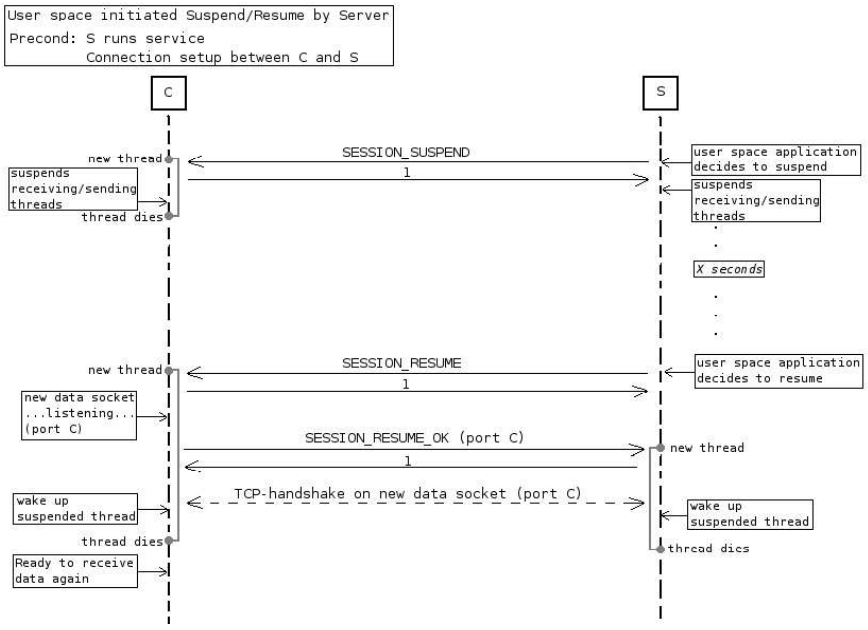

Figure 18: Suspend and resume procedure between Client and Server

P2P In figure 18 the suspend and resume procedure is displayed peer to peer, ie without the SLG.

The server's user application decides to suspend and sends a SESSION_SUSPEND message. It receives an ACK packet back whereafter both client and server suspends their threads. After an amount of time has passed when the server decides to resume the session it sends a SESSION_RESUME message and receives an ACK packet. The client sets up a new data socket and sends a SESSION_RESUME_OK to the server with the port number of the new data socket attached. The server sends the ACK packet back and connects to the client on the new data socket. The threads are awoken and the session is resumed.

4.4.3 Disconnection procedure

The disconnection procedure is commenced either by the server or the client. If the application is a file transfer program the server will initiate the disconnection procedure when all data is transferred. If the server, for some reason, would not initiate the disconnection procedure the client will initiate it when it has received all data. The figures shows the message passing with server as initiator.

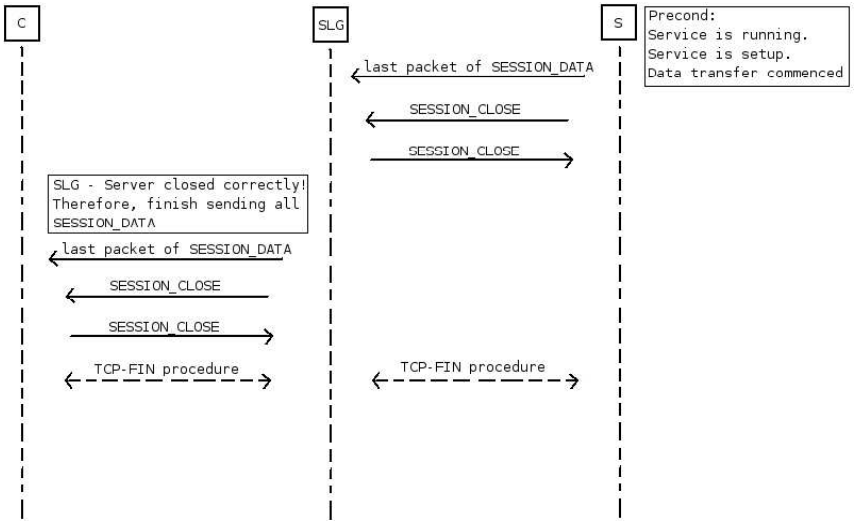

Figure 19: Disconnection procedure between Client and Server through SLG

with SLG In figure 19 the disconnection procedure between client and server through SLG is shown.

When the last SESSION_DATA packet is sent by the server it will send a SESSION_CLOSE packet and will receive a SESSION_CLOSE packet back as confirmation. After receiving the confirmation it will close the session. This is handled between the SLG and the server. Later when the TCP buffers are empty and all packets are passed on by the SLG it will send a SESSION_CLOSE packet to the client and expect one back until the session finally is closed.

It is a deliberate choice to close the session with one peer first and then with the other peer in order to simplify the procedure.

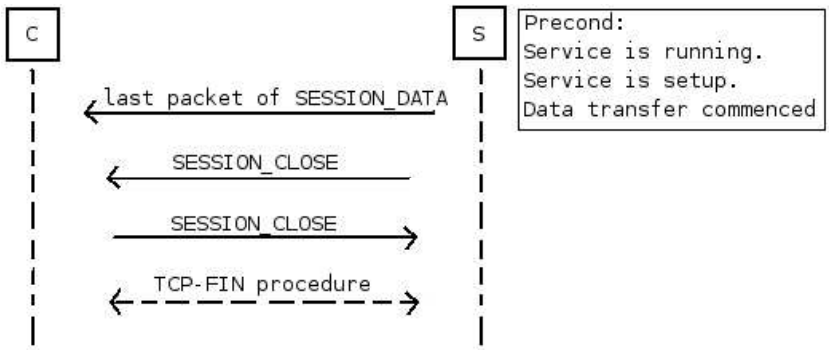

Figure 20: Disconnection procedure between Client and Server

P2P In figure 20 the disconnection procedure is shown peer to peer.

If it is done peer to peer, one peer sends a SESSION_CLOSE packet and the other one sends a SESSION_CLOSE packet back. The session is closed.

4.5 Name Resolution

The session layer uses a user space name resolution client while the rest of the session layer is written in kernel space. There need to be a way for the kernel to communicate to the user space client that it should lookup the ip address of a specific endpoint.

There were two suggestions how to achieve this where the first was to call user space functions from kernel space and the second to use signals. When researching whether it would be possible to call user space functions from kernel space we found out that this approach would not be possible and therefore chose to use signals from kernel space to user space.

4.5.1 Endpoint lookup

The name lookup sequence (figure 21) is initiated when the kernel function *session_sns_get_endpoint()* is called. It will first open a file */proc/session_sns* where it will write the endpoint name to lookup and thereafter send a signal to a waiting user space function. The kernel function will need to wait for a reply, because execution must be synchronous.

The user space function wakes up on the signal from kernel space. It will read the endpoint name from */proc/session_sns* and call the name lookup function and wait for a reply. When it receives a reply it will call the system call *sns_callback* with a pointer to the data structure. The user space function will pause until it receives the next signal.

The system call *sns_callback* will receive the pointer and pass this to *session_sns_get_endpoint()* which now can terminate.

4.5.2 Endpoint update

The endpoint update sequence works in the same way as the endpoint lookup apart from how it is initiated. It is started by a user space application which

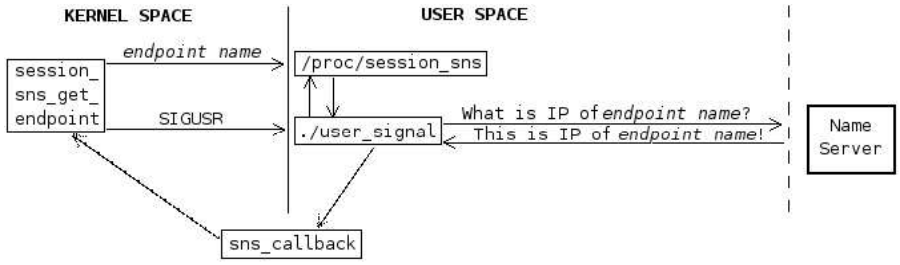

Figure 21: Endpoint lookup signaling

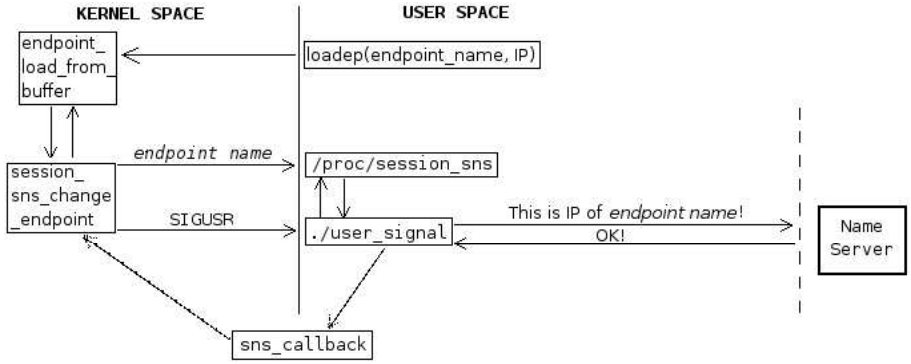

Figure 22: Endpoint update signaling

calls a kernel function passing the endpoint name and ip address. The ip address passed up to kernel space is used to create a socket for the endpoint to bind to. An endpoint is only allowed to accept connections on one ip type, ie either IPv4 or IPv6, although it might have both a public IPv4 address and an IPv6 address. Therefore the *user_signal()* needs to retrieve the ip addresses on its own.

There are different ways to retrieve a local ip address but not all of them will work unanimous together with the user space system.

First approach The first approach and the easiest way to retrieve the addresses was to use *getifaddrs()*[6] which unfortunately seemed to collide with the use of ioctl and socket in the name resolution client. This is a known bug with this function and as this did not work out a different approach was needed.

Second approach The second approach was to parse out the ip addresses as was done in [5]. This was proved to work but was complex as well as somewhat unreliable in regard to what actually came out of the function, an IPv4 address can be of different lengths eg 192.168.0.10 or 192.168.0.100. It is of vital importance that the end-of-line mark is set at the right place.

[6]http://www.hmug.org/man/3/getifaddrs.php

Chosen approach Both of the approaches above would have worked for both IPv4 and IPv6. The approach chosen are actually two different, one for IPv4 and one for IPv6.

The IPv4 address is retrieved from an ioctl call with SIOCGIFADDR as device request code. It returns the IPv4 address in a data structure. However, the data structure returned is not big enough to hold an IPv6 address which explains why this approach only works for IPv4.

The IPv6 address is retrieved from the proc filesystem. In the proc filesystem there is a file */proc/net/if_inet6* which contains the IPv6 addresses in a system: the localhost, link local addresses and the global address. The function reads from this file, parses out the global address and arranges it on the right form.

5 Analysis

In this part of the thesis the testing phase of the SLG is described. Many different tests have been performed and all of them a number of times to ensure that the SLG does what it is supposed to and can withstand for example malformed packets, attempts to resume non-existant sessions and attempts to setup sessions to non-existant endpoints.

5.1 Testing

Hardware The hardware used when testing was three machines and two usb sticks. The SLG was running on a desktop P3 machine, one host was running on a desktop P4 machine and the second host running on a P3 laptop. The name server was running on a fourth machine.

The SLG had two 100 MBit Ethernet cards, one connected to an IPv4 network and one connected to an IPv6 network. The desktop host also had two 100 MBit Ethernet cards connected to IPv4 and IPv6 while the laptop host had one 100 MBit Ethernet card connected to IPv4.

Software All of the machines were running Gentoo Linux with different versions of gcc. The session layer is implemented in the Linux kernel version 2.6.15 for both the hosts and for the SLG.

Testing These tests that have been performed cover both error handling and actual functionality. The SLG has been tested in parallell with the implementation of it and when it supposedly was fully implemented.

Packet support The packet support tests is to ensure that the SLG recognises all valid session packets and acts correct upon receiving one.

- Setup procedure (see section 4.4.1)
- Suspend procedure (see section 4.4.2)
- Resume procedure (see section 4.4.2)
- Disconnection procedure (see section 4.4.3)

Network support The network support test is to ensure that the SLG works on both IPv4 and IPv6. It should be able to forward packets from IPv4 to IPv4, from IPv6 to IPv6 and also between IPv4 and IPv6.

- IPv4 to IPv4
- IPv6 to IPv6
- IPv4 to IPv6
- IPv6 to IPv4

Network and Packet support together These tests ensures that it is possible to change ip address scheme and then resume using it.

- start on IPv4 and suspend, resume on IPv6
- start on IPv6 and suspend, resume on IPv4

Error tolerance at SLG It is important that the SLG does not crash when something goes wrong. It should be stable and be able to accept new connections after an error been dealt with.

- Endpoint unreachable
- Endpoint terminates prematurely without informing the SLG
- Endpoint tries to resume unrecognised session
- Malformed packets

5.2 Result

The test has been performed and shows that the SLG works and can handle all the different types of packets in the different scenarios. It is also fully functional on both IPv4 and IPv6 and can handle a switch of address scheme when resuming a previously suspended session. The error tolerance, as seen above, is also satisfactorily handled.

We conclude that the SLG has passed all tests.

6 Conclusions and future work

In this thesis I have described our approach to mobility. We do not see mobility as a problem but as a fact. Mobility is something we need to deal with, not something we should try to avoid. Our mobility management system provides mobility of individual communication sessions, sessions are no longer connected to a specific device and may be moved anywhere and at any time. The SLG provides mobility of address scheme for communication sessions setup through the SLG, it makes it possible to start a session in one ip network and then move it to another type of ip network.

Our concept of mobility is proved with the background material with references to todays research, through the design chapter explaining design choices and the implementation chapter and the analysis chapter together into a SLG prototype.

Bridging addressing schemes

With the SLG it is possible to communicate over different addressing schemes. IPv6 networks, public IPv4 networks and private IPv4 networks can all communicate with each other through a SLG.

Addressing scheme mobility

The SLG proves that it is possible to give addressing scheme mobility. An endpoint can start a connection in one type of addressing scheme and during a suspend and resume procedure change point of attachment to a different type of addressing scheme.

Peer-to-peer communication

While adding addressing scheme mobility and with an enhanced name server it is still possible to communicate peer-to-peer for session layers, if they wishes to do so.

6.1 Future work

For future work we state a couple of different ideas to continue the development of the SLG in particular and the session layer in general.

Transport Layer mobility

Addressing scheme mobility is performed at networking layer but there should be no reason to not being able to provide transport layer mobility as well. If changing the network layer protocol is possible, changing the transport layer protocol to an user preferred or an application preferred transport protocol should be no problem. See figure 5.

Session mobility

If an endpoint moves to a different network where another SLG is in use, the session state kept in the previous SLG should be possible to transfer to the new SLG. See section 3.2.10.

Naming scheme

A better naming scheme of endpoints. Now the meaning of an endpoint name is dubious and a naming scheme which makes an endpoint possible to differentiate from an e-mail address is needed. See section 3.3.1.

SLG discovery

When moving to another network you need to somehow get in contact with the SLG. Instead of using a static endpoint name of the SLG it could be managed through the use of DNS SRV [4]. See section 3.2.9.

SLG control port

We suggest a better way to contact a SLG than with a static port number. This should be possible to achieve with ICMP echo request/reply messages. See section 3.3.3.

References

[1] A. C. Snoeren, H. Balakrishnan. An End-to-End Approach to Host Mobility, MobiCom '00, 2000.

[2] A. C. Snoeren, H. Balakrishnan, M. F. Kasshoek. Reconsidering Internet Mobility, Workshop HotOS-VIII, may 2001.

[3] A. Conta, S. Deering. RFC 2463: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification, Dec. 1998.

[4] A. Gulbrandsen, P. Vixie, L. Esibov. RFC 2782: A DNS for specifying the location of services (DNS SRV), Feb. 2000.

[5] P. Arvidsson and M. Widell. Design of a session layer based system for endpoint mobility. Master's thesis, KTH, 2006. `ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/061005-Petter-Arvidsson-and-Micael-Widell.pdf`.

[6] B. Landfeldt, T. Larsson, Y. Ismailov, A. Seneviratne. SLM, A Framwork for Session Layer Mobility, Proceedings Eight International Conferene on Computer Communications and Networks, Oct. 1999.

[7] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, D. Spence. RFC 2903: Generic AAA Architecture, Aug. 2000. Updated by RFC3261 [19]. Status: PROPOSED STANDARD.

[8] D. C. Plummer. RFC 826: Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware, Nov. 1982.

[9] D Maltz, P. Bhagwat. MSOCKS: an architecture for transport layer mobility, IEEE - INFOCOM '98, Apr. 1998.

[10] S. F. E. Kohler, M. Handley. RFC 4340: Datagram Congestion Control Protocol, Mar. 2006.

[11] E. Nordmark, R. Gilligan. RFC 4213: Basic transition mechanisms for ipv6 hosts and routers, Oct. 2005.

[12] G. T. G. Albertengo, C. Pastrone. Moon: a new overlay network architecture for mobility and qossupport. *Next Generation Internet Networks, 2005*, pages 492–497, Apr. 2005.

[13] i-Technology News Desk. Linus Torvalds Outburst Sparks Fierce Debate: Does Open Source Software Need Specs?, Oct. 3, 2005. `http://fr.sys-con.com/read/136960.htm`.

[14] Information Sciences Institute. RFC 791: Internet Protocol, DARPA Internet Program, Protocol Specification, Sept. 1981.

[15] Information Sciences Institute. RFC 793: Transmission Control Protocol, DARPA Internet Program, Protocol Specification, Sept. 1981.

[16] International Organisation for Standardization. ISO standard 7498-1, "Information Processing Systems - OSI Reference Model. The Basic Model". http://www.acm.org/sigcomm/standards/iso_stds/\\OSI_MODEL/ISO_IEC_7%498-1.TXT.

[17] J. Postel. RFC 768: User Datagram Protocol, Aug. 1980.

[18] J. Postel. RFC 792: Internet Control Message Protocol, DARPA Internet Program, Protocol Specification, Sept. 1981.

[19] J. Solomon, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler. RFC 3261: SIP: Session Initiation Protocol, June 2002. Updates RFC2543 [20]. Status: PROPOSED STANDARD.

[20] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg. RFC 2543: SIP: Session Initiation Protocol, Mar. 1999. Updated by RFC3261 [19]. Status: PROPOSED STANDARD.

[21] C. Perkins. RFC 3344: IP Mobility Support for IPv4, Aug. 2002.

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1, June 1999.

[23] R. Finlayson, T. Mann, J. Mogul, M. Theimer. RFC 903: A Reverse Address Resolution Protocol, June 1984.

[24] R. Gilligan, S. Thomson, J. Bound, J. McCann, W. Stevens. RFC 3493: Basic Socket Interface Extension for IPv6, Feb. 2003.

[25] R. Hsieh, A. Seneviratne. Performance Analysis of Mobile IP and SLM, Proceedings Ninth IEEE International Conference on Networks, Oct. 2001.

[26] R. Moskowitz, P. Nikander. RFC 4423: Host Identity Protocol (HIP) Architecture, May 2006.

[27] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson. RFC 2960: Stream Control Transmission Protocol, Oct. 2000.

[28] S. Deering, R. Hinden. RFC 2460: Internet Protocol, Version 6 (IPv6) Specification, Dec. 1998.

[29] V. C. Zandy, B. P. Miller. Reliable Network Connections, MobiCom '02, 2002.

[30] W. Fenner. RFC 2236: Internet Group Management Protocol, Version 2, Nov. 1997.

[31] Wesly M. Eddy. At What Layer Does Mobility Belong?, IEEE Communications Magazine, Oct. 2004.

[32] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear. RFC 1918: Address Allocation for Private Internets, Feb. 1996.

A Dictionary

ACK Acknowledgement

Contents User defined object, for example a file.

Cyber-object A wider definition of contents than what fits in the endpoint concept

Endpoint An endpoint is in our context a directory with a name according to our naming scheme, eg bob@example.com can be the name of an endpoint.

ICMP Internet Control Message Protocol, used with ping.

IP Internet Protocol

- **IPv4** Internet Protocol version 4, the most widely used IP packet.
- **IPv6** Internet Protocol version 6, a relatively new IP packet type that slowly is being implemented and taken in use.

Mobility In our context: a move of any kind, anywhere, at any time. From our point of view mobility itself is not a problem, mobility will be accepted as a fact and not something we should try to avoid.

Mobility Management A system responding on mobility giving adequate reactions on any event taking place in the system.

NAT Network Address Translation

NS Name Server

Private IP address An ip address that is used in a private network. A private ip address is only unique within the private network and the ip addresses that are supposed to be used in private networks are defined by IANA [32].

Public IP address An ip address that is unique in the whole wide world.

Resume When an endpoint re-establishes its communication it resumes.

Service For example: a file server, web server etc

Session Communication session between two endpoints

Session Layer Our mobility management solution is implemented at Session Layer. It manages sessions including mobility of sessions, see [5].

Suspend When an endpoint pauses communication it suspends.

SLG Session Layer Gateway

TCP Transmission Control Protocol

UDP User Datagram Protocol

B Setting up the demo

The SLG is a gateway and as a gateway it has no function on its own. When setting up the demonstration three machines with the Gentoo Linux base system installed are required.

Precondition Name Server is running.

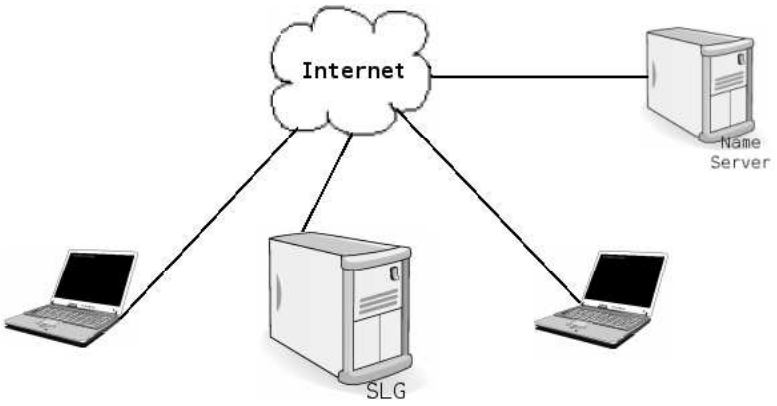

Figure 23: Demo Setup

B.1 Installation

Instead of downloading the source from the gentoo repositories the session layer enabled sources are kept in the local cvs. The module "kernel-gw-cs" must be installed on the hosts and the module "kernel-gw" must be installed on the SLG. With the sources there are two ".config" files, one for use with user-mode-linux and one for use on a real machine.

After compiling the kernel and rebooting the machines the user-space applications must also be installed. For hosts the module "user-gw-cs" is used and for the SLG the module "user-gw". They both come with README-files and install scripts. When they are installed the installation part of the setup is completed.

B.2 Setup

The program that is used to display the capabilities of the session layer is a file transfer program. It sends a file from one endpoint to another endpoint.

An endpoint is the name of a folder, for example "test@example.com". In this folder there must be a file ".endpoint". An endpoint can reside on a machine but also on a usb stick. If the endpoint resides on the machine it must be added in the "/etc/conf.d/local.start" file. The password which is used in communication with the name server must be added in the "/etc/session_layer/endpoints/local_endpoints.txt" file, there is an example file called "local_endpoints.txt.example". If the endpoints resides on a usb stick it *must not* be added in "/etc/conf.d/local.start".

Installation should have installed fluxbox with a session layer menu. In the menu there are shortcuts to start server and client, make sure that the shortcuts links to the correct endpoints.

It is also important to be certain that the name server is updated with correct address information, type “source /etc/conf.d/local.start”.

B.3 Test

Installation and setup should now be completed and it should soon be possible to test the session layer capabilites.

Important to notice is that in the server program the file is called “testfile”, this file should be in the endpoint directory of the server. The client program needs to now in advance how big the file being transferred is. After changing the filesize in the client program, type “make install” in the test directory.

This test is for use with endpoints on usb sticks. Start the server on one machine by clicking “start storage” and then start the client on the other machine by clicking “start client”. The transfer should have started. If you want to suspend the communication click on “saveep” for client or server in the fluxbox menu. Then the transfer is suspended and the usb stick is unmounted, it is now possible to remove the usb stick. When you later plug it in it will automatically after a few seconds resume the previously suspended transfer.

The demo setup in short

SLG Installation

- cvs checkout “kernel-gw”
- cp config-070301 .config
- make && make modules_install
- reboot
- cvs checkout “user-gw”
- .\install.sh

Setup

- create endpoint - mkdir slg@verkstad.net
- cd slg@verkstad.net
- touch .endpoint
- Add it in “/etc/conf.d/local.start”
- Add endpoint name and password in “/etc/session_layer/endpoints/local_endpoints.txt”
- source “/etc/conf.d/local.start”

Hosts Installation

- cvs checkout “kernel-gw-cs”
- cp config-070301 .config
- make && make modules_install
- reboot
- cvs checkout “user-gw-cs”
- .\install.sh

Setup

- create endpoint on usb stick - mkdir test@example.com
- cd test@example.com
- touch .endpoint
- Add endpoint name and password in "/etc/session_layer/endpoints/local_endpoints.txt"
- Make sure fluxbox shortcuts links to correct endpoints
- source "/etc/conf.d/local.start"
- Repeat this procedure on the other machine (with a different endpoint name of course)

Test

- Add the file, testfile, to be transferred in the server endpoint directory
- Set the size of the file to be transferred in the client program "user-gw-cs/session-lib/test/client.c"
- In the test directory - make install
- Start server on one machine
- Start client on the other machine