Diplomarbeit zum Thema:

The P2P-Oracle Server: Implementation and Performance Evaluation

Fakultät IV – Elektrotechnik und Informatik
Intelligent Networks / Intelligente Netze (INET)
Research Group of Prof. Anja Feldmann, Ph.D.

Ingmar Poese
May 10, 2009

Prüfer: Prof. Anja Feldmann, Ph. D.
Betreuer: Obi Akonjang

Die selbständige und eigenhändige Anfertigung versichere ich an Eides Statt.

Berlin, den May 10, 2009

Ingmar Poese

Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit dem Konzept und der programmiertechnischen Umsetzung des Oracle Server. Der Oracle dient dazu IP Adressen nach einer beliebigen, meistens in Relation zur Entfernung der Quelle stehenden Metrik zu sortieren. Mittels dieser Technik ist es möglich, dass Netzwerke, die auf dem Internet aufbauen, ihre Struktur besser der Topologie der Internet Provider anpassen können ohne diese explizit zu kennen. Mit Hilfe dieser Technik können Internet Service Provider(ISP) Nutzern Vorschläge unterbreiten, welche Verbindungen sie bevorzugen sollten. Durch diese Vorschläge erhalten die Netzwerkinhaber die Kontrolle über die Datenflüsse in ihren Netzwerken zurück und können diese besser und effektiver lenken, wodurch auch die Effektivität für den Benutzer steigt.

Um seine Aufgabe zu erfüllen muss der Oracle Server die detaillierte Sicht auf das komplette Netzwerk der ISP haben. Für eine gute Skalierbarkeit und optimale Ressourcenausnutzung ist ein Multi-Thread-Ansatz eine wichtige Vorraussetzung. Wie dieser entworfen und implementiert wurde, ist ebenfalls Bestandteil der vorliegenden Arbeit.

Die Diskussion über die internen Mechanismen und die Umsetzung stellt die für den Oracle Server entwickelten Objekte und ihre Zusammenarbeit vor. Daneben wird auf das Datenmodell des Oracle Servers, die Austauschbarkeit der Gütefunktion und die Funktionsweise des Testclients eingegangen.

Mit Hilfe des Clients werden Testläufe für den Server entwickelt, die den entwickelten Versuchsaufbau durchlaufen. Diese testen den Server auf die maximale Antwortrate, welche Zeit benötigt wird bis eine Antwort zum Client zurückkommt und wie sich die entwickelte Software unter denial-of-service (DoS) Attacken verhält. Die Ergebnisse dieser Tests zeigen, dass der entwickelte Server bis zu 2,5 Millionen Anfragen pro Minute bei einer mittleren Antwortzeit von 2,6 ms beantworten kann.

Abstract

The Oracle is an ISP-provided decision facilitator service. Its uses are multifold and include the ranking of IP addresses and prefixes based on customized criteria such as the network distance between the source of a query and multiple candidate peers. This service assists peers in overlay networks to better select their neighbours and sources from which to download content. It also enables ISPs to reduce congestion within their network and transit links, without revealing any topological or performance information.

In this thesis, we present the design, prototypical implementation and analyses of the Oracle service, with respect to the primary goals of efficiency, scalability and flexibility. The contributions of the thesis are three-fold. First, the disciplines underpinning the architecture of the Oracle server are outlined. Second, efficient data structures and multi-threaded programming techniques used to boost the performance of the Oracle server are presented. Third, an exhaustive performance evaluation study of the implemented Oracle server is provided.

The experimental results discussed in this thesis are supported by extensive system and performance analyses which provide evidence in support of the efficiency and scalability of the proposed architecture. To further demonstrate its flexibility, a series of experiments involving the prototype server and real clients are also presented.

Contents

1 Introduction **3**
1.1 Routing and traffic engineering . . . 3
1.2 Overlays . . . 3
1.3 Underlay-Overlay interaction . . . 4

2 The Oracle principle **5**
2.1 Benefits of an Oracle aided neighbourhood selection . . . 5
2.2 Oracle in an ISP-network . . . 5
2.3 Cost for running an Oracle server . . . 6
2.4 Related work . . . 6

3 Implementation issues **8**
3.1 CPU usage observation . . . 8
3.2 Memory restraints . . . 8
3.2.1 Memory requirements for routing tables and path caches . . . 9
3.2.2 Subnet aggregation . . . 10
3.2.3 Routing abstraction . . . 10
3.3 Multithreading . . . 12

4 Implementation of an Oracle server **14**
4.1 Choice of programming language . . . 14
4.2 Development tools . . . 14
4.3 Data types . . . 15
4.4 Constants . . . 15
4.5 Internal object structure . . . 16
4.5.1 Router . . . 17
4.5.2 Prefixmatch . . . 17
4.5.3 Networklist . . . 18
4.5.4 Network . . . 18
4.5.5 Oracle . . . 21
4.6 Configuration files . . . 22
4.7 The console . . . 22
4.8 Socket handling . . . 24
4.8.1 The proxidor protocol headers . . . 24
4.9 Additional server modes . . . 28
4.9.1 Query client mode . . . 28

5 Server performance test setup **29**
5.1 Test setup . . . 29
5.2 Test network . . . 29
5.3 Test composition . . . 30
5.3.1 Performace Test Composition . . . 32
5.3.2 Denial of service tests . . . 33
5.4 Additional Tests . . . 33

6 Test results **35**
6.1 Regulated clients . . . 35
6.2 Replies per second . . . 35
6.2.1 Packet loss . . . 37
6.3 Round-Trip-Times . . . 38
6.4 Varying request sizes . . . 39

6.4.1 Replies per second . . . 39
6.4.2 Round-Trip-Times . . . 40
6.5 Flooding behaviour . . . 41
6.6 Small vs. big internal network . . . 43
6.7 Stability . . . 44
6.8 Test result summary . . . 45

7 Conclusion **47**
7.1 Future Work . . . 47

1 Introduction

The Internet is a network of interconnected autonomous systems (AS). This implies it is controlled by multiple network providers, ranging from local to global Internet service providers (ISP). Every AS has explicit knowledge of its own network topology, yet knows little about the setup of others.

The Border Gateway Protocol (BGP) is used to announce subnet information to other networks. This includes information about the local subnets, as well as advertising updates from third parties. Relaying the merged information to neighbours creates a chain of updates reaching every AS connected to the global system of networks. BGP is deployed on the links between different AS.

In order to reach the subnets announced by network providers, connections are established through multiple AS'. This generates transit traffic along the path of connections through different ISPs. Each time the connection enters a new network, contracts between the neighbouring AS state the transit cost for using the link. The expenses for conveying connections are directly related to the size of the transferred data. It follows that keeping traffic local reduces transit costs, therefore reducing the ISP's expenses.

However, an AS usually has multiple links to many other AS'; each with their own individual contracts, policies and link properties. This leaves the question of where transit traffic is to be sent, as a subnet can be reached over several links. At this point it becomes beneficial for a network provider to have control over the traffic it generates and that which transits its network. Directing connections to subnets via low cost or free links enables an ISP to optimize its expenses regarding transit costs.

1.1 Routing and traffic engineering

Each router in a network has its own routing table. It contains the subnets attached to its own AS merged with the received subnet declarations announced via BGP from neighbouring AS'. A connection through the Internet is based on the individual local forwarding decision made by every router along the path.

In the case of a subnet being announced via multiple border gateways, an AS can implement traffic engineering to modify the routing tables, ensuring a minimum expense for transiting traffic to subnets outside of its network borders.

However, the capacity of network links is limited. Engineering the flow of packets in the network to minimize the operational cost for transiting traffic leads to an increase in latency and congestion on the utilized network links. Along with the degrading quality of the network comes a deterioration of quality of the Internet connection to the customers. Thus, ISPs need to minimize the operational expenses of running their networks while supplying customers with high performance Internet connections.

One prerequisite of traffic engineering is the knowledge of the traffic-flow into, through and out of the network being optimized. As it is impossible to infer and react to the exact stress on the topology in real time, the network load is predicted using previously recorded measurements. An ISP therefore needs to apply traffic engineering at regular intervals to react to changes in the network load.
New applications using the Internet as a basis for communication are deployed in unpredictable ways. Each introduced application changes the shape of the traffic an ISP is experiencing, thus triggering the need for traffic engineerning anew.

1.2 Overlays

Overlays are virtual networks running in the application layer. They build their topology by creating virtual links on top of that of the Internet. Connections being established in the overlay network can take a different path compared to a direct communication taking place in the physical layer.

Skype[12] is an example of an application offering this kind of service. In the case where two hosts running Skype are incapable of establishing a direct connection, a relay is used to forward the connection between the two. This relay is chosen arbitrarily from third parties on the Skype network unaffiliated

with the two hosts wishing to communicate. The only requirement for this intermediary is connectivity to both hosts.
In recent years, overlay networks have demanded an increasing share of the Internet's bandwith. They offer a wide range of services enriching the customer's Internet experience. Overlays are unaware of the underlying network topology and AS borders since they see the entire Internet as one global network.

ISPs have no control over the routing implemented by overlay networks. This, together with the application layer routing these types of services introduce, produces difficulties regarding the ISPs traffic engineering. Contributing to this point is the dependency of the traffic on the Overlay's topology. Content sharing peer-to-peer(p2p) networks raise new challanges, as they distribute large amounts of data[13].

Here the ISP runs into a dilemma. Overlay networks provide a service to customers they need to support. Additionally, ISPs directly benefit from overlay networks, as they spur the customer's bandwidth demands. However, overlays produce higher transit costs for ISPs, as they are unable to engineer their own traffic as easily.
Overlays use sophisticated methods to optimize the user experience[7] including active probing to determine the quality of virtual links. However, probing requires bandwidth, creating congestion in the physical topology. In contrast, ISPs are familiar with the underlying network topology, enabling them to supply superior information to that which the overlay networks infer through active measurements.

1.3 Underlay-Overlay interaction

The Oracle service is proposed as a solution to the problems introduced by overlay networks regarding traffic engineering of ISPs. Through the Oracle server, the overlay clients acquire suggestions about the quality of destinations, without having to probe virtual links. It also allows the ISP to gain some control over the neighbourhood selection in overlay networks, thus facilitating improved traffic engineering, while making it unnecessary to reveal the network topology. The ISP cannot, however, force the overlay clients to use the offered Oracle service.

Offering customers a better overlay network performance when inferring connection information from the Oracle service ensures that it will be utilized. Using the Oracle to optimizie the overlay enables the ISP to better engineer traffic within its own network and lower its operational expenses. Customers can therefore expect an increased performance while running overlay applications.

2 The Oracle principle

The Oracle is a service offered by an ISP to its customers. Its primary function is to help order IP addresses relative to each other based on their position on the Internet. For this, a list of potential candidates is sent to the Oracle server. Upon receiving a request, the Oracle ranks the list of potential peers to the best of its knowledge. This scheme helps peers to quickly and effectivly find appropriate neighbours, enable streaming clients to aquire optimal sources, maximize the performance of overlay networks and enable improved traffic engeneering for ISPs .

The Oracle shares similarities with DNS servers. They convert human readable Internet addresses into IPs so that the clients can connect to the appropriate server. In the same way, the Oracle assists peers when choosing from among multiple candidates. They both need to be fast, flexible and scalable. Also, both services are not essential for the Internet, but make it more convenient to use.

2.1 Benefits of an Oracle aided neighbourhood selection

In most Overlay networks on the Internet, the neighbours are chosen randomly. While this complexity is a practical approach, as it keeps the networks clustering coefficient high and helps them to stay connected if clients suddenly go down, it is bad for an ISP since they cannot influence what client connects to which one as their next neighbour. This leads to inefficient use for the user and unnecessary cost for the ISP[7].

When a client tries to join an overlay, it gets a list of potential neighbours from defined sources on the Internet that is held at a central place. Usually this list of potential candidates is randomly compiled. The client then chooses how many neighbours it needs and connects to those. Many overlays perform measurements to determine which clients are best to connect to. This is, from the ISPs point of view, even worse, as it increases the unneeded traffic and makes sure that almost all of the possible neighbours are contacted.

The same principle is applicable when choosing from among multiple download sources. While a search for content that a user might want runs over multiple hops on the overlay network, the download connection for the content is initiated directly from the client to the source. In most overlay networks, the required content is offered at several locations spread over the Internet. As soon as the client has a choice of different sources to download from, it is not clear how it decides what source is the best. Especially in networks where content is spreading fast, this becomes an issue, as the required data can be downloaded from a lot of potential candidates.

In both cases, the Oracle can recommend the best choice to the client. For this to happen, the list of potential candidates must first be sent to the Oracle server. It can help determine the best neighbours or sources to connect to. This choice can be best made by the Oracle, because it has extensive knowledge of the underlying network structure and properties. Therefore, it can match the possible connections and figure out which of the possible sources should be prefered and which should not.

This will increase the performance and user experience with the overlay network as well as give some level of control to the ISP over the traffic within their own network[1].

2.2 Oracle in an ISP-network

An ISP which runs the Oracle will gain the benefit of being able to take back some control over the overlay networks that are currently running on top of their infrastructure. The Oracle is first loaded with information about the entire network infrastructure. This includes metrics that the ISP would not give out to third parties or make public. The information can contain the connections between routers, the bandwidth and latency of all known links within the network, the peering points of the AS with other ISPs and the subnets that can be reached over them, as well as where clients are and what their up- and downstream is. All this information is already stored somewhere at the ISP and only needs to be supplied to the Oracle.

Based on these metrics and a custom-made ranking function, the Oracle server can start serving clients. The offered service ranks lists of potential connection candidates without giving out specific

information about the underlying network infrastructure. Using this service, the overlay clients no longer have to measure connection characteristics and the ISP can also influence how the overlays are built.

However, clients on the overlay network are not forced to use the service offered by the Oracle. Thus, they need a reason to use it and to keep using it. The only thing an ISP can offer its users as a reason to use the Oracle, is performance. If it increases while the Oracle is used, and reduces if it is not utilized, then customers will stick to querying the Oracle for information. Therefore, it is important for an ISP offering this service not only to minimize costs but also to improve end-user experience.

2.3 Cost for running an Oracle server

Running an Oracle server only needs hardware as well as information about the network structure it is serving. The topology information can either be supplied manually or can be taken directly from the internal routing protocol that is running within the network core of the AS. Both methods enable the Oracle to be able to rank lists of potential connection destinations based on network metrics. Also, the administrators of the network can supply a custom function for the ranking to optimize the Oracle to their need if the built in function is not performing well.

Once this has been set up, the ISP only needs to publish the IP or host name of the Oracle server and customers can immediately start querying it. As the Oracle does not give out any information about the network itself, but only reorder the queries sent to it, the ISP does not leak any information about its network topology.

The cost for running an Oracle server is minimal compared to the potential gain that an ISP could get from implementing it.

2.4 Related work

Along with the Oracle, there have been other ideas proposed to handle the growing Inter-AS traffic generated by overlay networks. An effort related to the Oracle scheme is the P4P approach [16]. It proposes that the topology information for the Overlay networks is hosted by a third party. This third party gets topology information from multiple or, in the best case, all AS on the Internet and combines them into one large network with known network boundaries. This service is then used by overlay clients to distinguish their position relative to the potential sources.

While this solution has a better view on the topology as the Oracle has, it poses the problem that the ISPs need to reveal their network infrastructure. Even if this third party is not giving out any topology information, this is problematic, as ISPs are very reluctant to supply any information about their network structure. Another concern is that the benefit for a single ISP cannot be seen or controlled directly by itself, but is in the hands of a third party.

Another approach to handling the behaviour of the overlay network is by using content distribution networks (CDNs) to infer the proximity of peers[4]. This solution uses the dynamic redirection of the CDNs and infers that if two clients are sent to the same server, they must be close to each other, implying that they should also be connected.

This approach does not need any network structure knowledge or extra server running within an ISP to work. It builds entirely on an already existing system that is distributed on the Internet. However, as the CDNs have no explicit knowledge about the network structure of the ISPs, it is possible to attain locality with it but it does not guarantee cost reduction for the ISP. If two peers, according to the CDN, are close to each other, it does not mean that they are in the same ISP. There can still be multiple ISP boundaries between them costing a lot of money. Thus, this whole scheme is based on the accuracy and distribution of the CDNs. Again, the ISP has no control over the Overlay networks and is not involved in its shaping.

A more specific approach for Bittorrent is to modify the client to use general topology information[3]. Assigned AS numbers are generally known even to outsiders of ISP networks and can be mapped to clients. ISPs can also mark the Bittorrent packets as they pass through their proxies or border gateways

into other networks to give an idea of where the packets have been. However, this does not allow ISPs to shape the traffic in their network, but only what leaves it. It is also only proposed for the Bittorrent network which might use a fair amount of all overlay traffic but leaves other networks unattended.

As Bittorrent is securing its services against outsiders trying to analyze what is being sent, it tends to encrypt its traffic. This in itself would not pose a problem if it was only Bittorrent that was encrypting their traffic. But if multiple overlay networks start to do this, they are no longer distinguishable from each other by the ISP, making it hard to find the right markers or classification for the Bittorrent traffic. Also the proposed packet manipulation for marking the Bittorrent packets would not work anymore.

One proposition that is not utilizing neighbourhood selection, but attending to the content of the overlay network, is caching of P2P-traffic [11]. it means implementing a service run by the ISP that listens in on traffic on the overlay network and caches the content that is being distributed. If a second client is to request the same content, it can be supplied by the cache. This reduces the need for the traffic to leave the ISP. Furthermore, the ISP would have complete control over the cached traffic and can position the service at a good point in their network with enough resources available to serve the requests.

The major problem of this scheme is that if an ISP caches data and then redistributes it via the cache, it might become an accessory to breaking copyright law. The content that is shared over the P2P network cannot be automatically validated by the ISP to be legal. Thus, sharing it might lead to legal issues. Another point that has to be taken into consideration is that this scheme also suggests that the P2P traffic be unencrypted and can thus be monitored. Once the communication between the clients is encrypted, the cache will no longer work.

3 Implementation issues

This Section deals with the development considerations regarding the implementation of an Oracle server. It will discuss theoretical approaches used to get the server running at a sufficient speed while staying memory efficient.

3.1 CPU usage observation

The implemented Oracle server needs to be fast and scalable at handling requests. These are the key criteria for running an Oracle server. Thus, the approach taken to implement an Oracle server must be independent of the following;

1. **Number of handled routers**
   In the Oracle internal database, routers are looked up frequently to determine where source and destination networks of connections are. Finding the router a subnet is attached to must be a constant operation and must not be dependent on the amount of routers in the database.

2. **Path properties from source to destination**
   When the source and destination of a connection have been found, the CPU time concerning the retrieval of path properties between these two must not be dependent on the length of the path. The calculation of this information between source and destination must be constant regardless of the distance between the two.

3. **Number of known subnets/routes on the Internet**
   The number of subnets on the Internet known to the Oracle grows constantly as well as fluctuates. Therefore, an approach is needed that will make sure that even if the routing tables of the routers continue to change, the Oracle server will not degrade in speed when subnets are appearing or changing on the Internet.

Requirement 1 can be solved by organizing the routers in a hash table. This will ensure that the speed for lookups is the same no matter how many routers are known to the Oracle.

Having a constant lookup speed, regardless of the path length between two routers means caching all known paths to fulfil requirement 2. Using the cache to get a constant speed on the path lookup will also increase the memory needs of the Oracle.

The prefixes for the known networks from requirement 3 will be organized in a binary tree. This does not ensure a $O(1)$ operation, but will supply a $O(\log(n))$ lookup speed. Also, the maximum depth of the tree is 32 for IPv4 addresses.

As all three conditions can be met, it is possible to have an implementation of the Oracle server suited for any size ISP run without any speed degeneration regarding its internal network setup or size.

3.2 Memory restraints

Since all requirements from 3.1 for the Oracle server can be met with standard techniques, the question is how to implement them in an effective way concerning memory. On the one hand, the number of routers within the Oracle network is not much of a concern, as their number should not grow uncontrollably. Even in large networks, this is still a number in the thousands which might use up a few hundred megabytes of Memory.

On the other hand, the path calculation as well as the subnets on the Internet change rapidly and without the direct interaction of the AS is served by the Oracle. Current measures show that today Internet backbone routers hold up to 250,000 routes[10]. Also, the path between the source and destination,

as discussed in 3.1 requirement 2, needs to be pre-calculated and stored in order to ensure the lookup performed on the database is running at a sufficient speed. This, in its worst case, would mean holding every path from every possible source subnet to every possible destination subnet in the database of the Oracle Server.

3.2.1 Memory requirements for routing tables and path caches

A router forwards a packet based on its own routing table. It does not know nor does it care where the packet came from and it is not interested in what the next router will do with the packet. Thus, for every known subnet to the router, there is one route attached to it which holds the subnet and a next hop. If this next hop is missing, the router sends it towards its default gateway. In case there is no default route, the router will usually reply with a message saying that there is no route to the network.

In contrast to a router, the Oracle server is not interested in the local forwarding of one hop but in the path the connection takes through a network and ultimately through the Internet. To calculate this path correctly, it needs to know the routing tables of the routers along to the path. Therefore the Oracle needs the routing table of every router within the network it serves to simulate the flow of the connection through its network.

Assuming the worst case, the amount of routes the Oracle needs to store can be calculated by the amount of routers the server knows multiplied by the number of routes every router has. In the case of equation 1 the number of known routes to subnets is the maximum known to a router. Therefore, this is the worst case scenario, as routers at different positions in the network are able to aggregate subnets to reduce the size of the routing tables.

$$total_cache_entries = number_of_Routers * number_of_known_subnets. \quad (1)$$

ISP networks today have 1000 routers or more working their network. As memory is neither expensive nor hard to obtain anymore, it should not be a major concern to handle the needed requirements. But the amount of memory needed to hold 250,000 routes in 1000 routers is overwhelming, as every IPv4 route needs at least six bytes of memory. These six bytes hold the actual IP-Address (four bytes), the subnet definition (one byte) and the interface the packet should exit on (one byte). This scheme only works if there are no more than 256 network interfaces on each router and if every connection between two routers is a point-to-point connection. If there are broadcast networks attached to a router, a gateway needs to be specified to forward the packet, which would mean that four bytes instead of one are needed for the network interface.

The explained memory scheme only works for IPv4 networks. In case IPv6 addresses are being used, the amount of memory would increase by another twelve bytes per route, as the IPv6 addresses have 16 instead of four bytes. All other conditions for the IPv4 networks also apply for the IPv6.

This means, in the case of using 250,000 routes per router and having 1000 routers handled by the Oracle, that 1.4 gigabytes of Memory would be needed to store just the routing tables of the routers. In the case that the Oracle was to support IPv6 as well, the memory requirement would expand to 4.1 gigabytes of memory. Figure 1 shows the Memory requirements of the Oracle for an IPv4 Network stored in its database with this scheme.

This calculation uses the ideal case that no memory is used for managing the routes, as well as the assumption that the memory does not need to be partitioned into blocks, but can be directly accessed on a byte basis. Usually this assumption does not hold, as 32-bit machines can only address memory blocks of a size which is a multiple of four. In this case, another two bytes would be lost, since the machine can only address eight bytes instead of six. This would increase the memory requirements yet again.

Another point that will also increase the memory consumption is the fact that with every route to a subnet in a router, there is the need to also save the entire path information with it to still be able to comply with the cache requirement as defined in section 3.1. The size of the path information is not known in advance, but can be estimated around 10 to 20 bytes which are added to the already existing six Bytes for IPv4 routes. This additional memory can be spent for storing the amount of hops, the

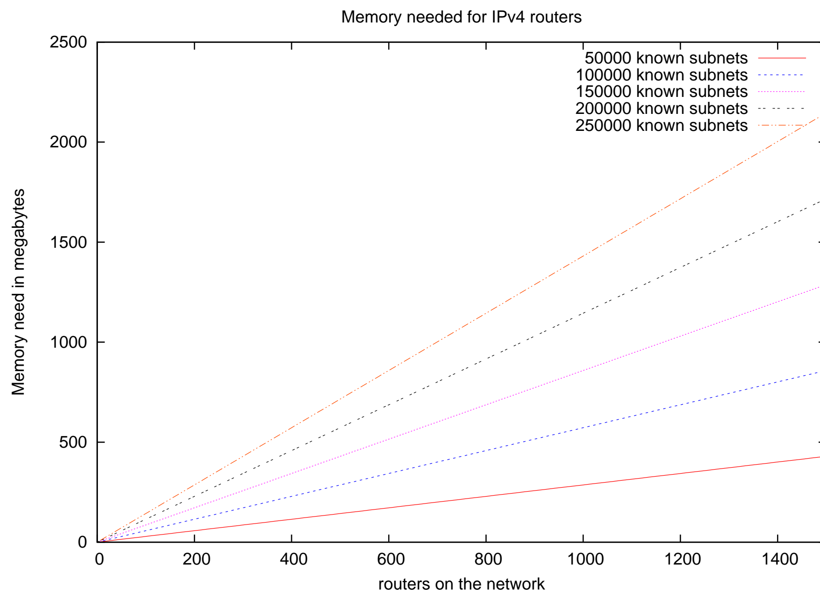

Figure 1: Memory needed to store conventional routing tables of IPv4 Routers in the Oracle Database

minimal bandwidth along the path, the aggregated latency of the connection or the AS distance of the network from the border gateway router.

Therefore, it is safe to assume a size of 20 to 30 bytes per route including the caching of the path as well as memory segmentation. Figure 2 shows the estimated memory consumption using 24 bytes per IPv4 route where 6 bytes are used for the route itself and 18 bytes are spent on the path cache.

3.2.2 Subnet aggregation

If a router has two subnets that can be aggregated into one, it will do this. This works because a router does not care about what the next router does with the packet. In contrast, the Oracle cannot do this aggregation within its database of the network, as it needs to deal with the path between two points in the Internet and not a local forwarding decision.

The Oracle server is in a position where it must hold the path from any of its routers to any known subnet found in the Internet in its memory at all times in order to still comply with requirement 2 from chapter 3.1. But if a router has an aggregated subnet which splits along the path within its own network, it has to store two cache entries for the same subnet in the router. Along with the two cache entries, it also needs to save which path information belongs to which subnet. Therefore, it essentially splits the aggregated route into two more specific ones. It is even possible for more than one split to occur on the path, causing more routes to be present.

Therefore an Oracle server will not be able to perform route aggregation to optimize its subnet lookups. What's more, it may need to split up routes that are aggregated on the real routers in the real network in order to be able to have a properly filled cache for fast lookups.

3.2.3 Routing abstraction

In order to avoid using vast amounts of memory to handle the routing tables as well as the path cache, a layer of abstraction to reduce the saved path is needed in the Oracle. For this to work, the network known to the Oracle is split into two categories. One category is the internal network. This part only consists of routers and links that are directly known to the Oracle server. Furthermore, the routing tables

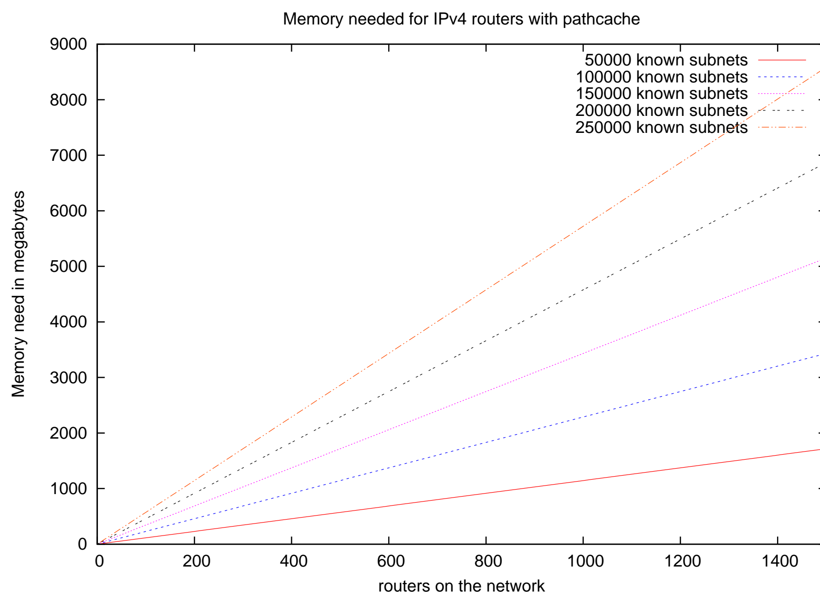

Figure 2: Memory needed to store routing table with path cache of IPv4 Routers

of the routers in this part of the network are reduced to only known routes to destinations in the same network. This effectively reduces the amount of routes per router to the routers in the same network.

The other category is subnet information. This covers all prefixes known to the Oracle, no matter if they are attached to a known router in the internal network or are being announced by a different AS. Basically, the whole knowledge of all known subnets throughout the Internet are declared to be external networks.

Once this abstraction is put into effect, the subnets need to be saved only as often as they are known to different exit routers. An exit router is a machine that, from the Oracle's point of view, still knows where a subnet is, but is the last hop within the network that is managed by the Oracle. These are either border gateway routers, as they are the transit point into someone else's network, or source routers in the internal network where the actual subnet is attached. It is important to mention that exit routers are also internal routers, as they still belong to the network managed by the Oracle server.

By separating these two kinds of networks, it is possible to drop the routing table size of internal routers to the total amount of routers in the network itself. This is due to the fact that the path is not known to the Oracle once it leaves the internal network and is aggregated in its database anyway. Therefore, a path now consists of two different sub-paths. One is the known path through the internal network itself. This is the one that also needs to be cached. The other part is the one outside of the Oracles network which it can neither see nor determine in detail. This subnet information comes already aggregated to the Oracle via BGP updates from other AS. So instead of 250,000 possible routes within each router, it needs only save 1000, if the internal network has 1000 routers.

The space of memory needed to store the path information has not decreased from the calculation done in section 3.2.1 but the amount of routes and cached entries that need to be saved has dropped drastically. Now every router must save the path information to every other router, while the external subnets only need to be saved as often as they have different exit points. The cache entry still has a size of about 30 to 40 bytes to store all the needed information, as well as another 30 to 40 bytes to store each external subnet on each exit router. But even with 250,000 known subnets, each having three exit points and 1000 routers within the internal network, the memory need will be below 150 Megabytes, as shown in figure 3. Equation 2 shows the formula to calculate the cache entries needed to be stored within the

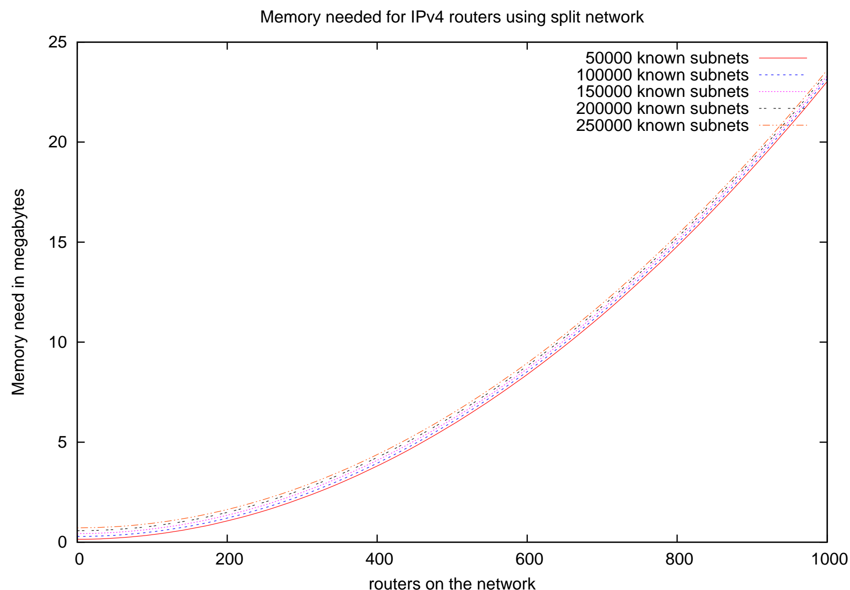

Figure 3: Seperated network memory usage with three exit points per external network

database of the Oracle if this abstraction is used.

$$total_cache_entries = (number_of_routers)^2 + average_exit_points * external_subnets \quad (2)$$

Since explicit routes to subnets on the Internet are now missing from the routers, it is not possible for them to decide which exit point should be taken, as they are not able to figure out the exact routes at every hop anymore. To solve this shortcoming, every link that exists between two routers is assigned a weight. These weights can be taken directly from the internal routing protocol and can be used to locate the best exit point within the network the Oracle servers.

Furthermore, all internal paths are being pre-cached so the path-finding between a source and a destination is still a constant operation. But since multiple exit points can exist, the path finding has to look at every one to figure out which has the lowest internal weight assigned.

In addition to finding the best exit point, there is the overhead of adding the external network characteristics to the internal path to gain the full path information. Again, this is a constant operation in CPU time, but one that was not needed before the routing abstraction. However, the overhead is insignificant compared to the memory being saved.

3.3 Multithreading

One more issue that needs to be dealt with before the implementation is multithreading. Today, most CPUs have multiple cores, each being a processor. If the Oracle server was single-threaded, it would not be able to use multiple cores. Therefore, having a scalable Oracle calls for multithreading.

Having multiple threads means there is no certainty within the flow of instructions in the code. Instead, events can occur asynchronously without any chance to predict them. As long as no data is modified, it does not matter where threads read and if they accidentally read the same data. But if any thread needs to write to the memory where another thread could potentially read at the same time, it could lead to data inconsistency. In the worst case, this inconsistency can crash the entire server.

In order to prevent such a crash, it is necessary to synchronize the threads with one another. To accomplish this, the server splits itself into two different kinds of threads. The first kind is called *worker*

thread and is only allowed to read data. Since it cannot modify anything, it is impossible for this kind of thread to cause any data inconsistency. There can be any number of worker threads as needed running parallel without interfering with each other. These threads are used to answer queries which are sent to the server.

The other kind of thread is called the *modification thread*. This type is allowed to modify data within certain boundaries. For one, it must always synchronize the worker threads if something needs to be changed within the database of the Oracle.

The synchronization is done by checkpoints. If a worker thread is currently executing between the two checkpoints, it is considered safe - what means it is currently not accessing any data in the database. What the thread is doing between the checkpoints does not matter, as long as it is known to be between them. To achieve a safe change within the database of the Oracle server, the modification thread can lock the exit checkpoints of the worker threads, meaning that any worker thread that wants to exit the safe area via the checkpoint is blocked until the modification threads allows it to continue. The modification thread waits until all threads are between the checkpoints. Once this is the case, the modification can take place, as no other thread is now able to read the database. After the alteration is complete, the lock on the exit checkpoint is lifted and the worker threads are allowed to resume their work.

This scheme poses the problem that while a modification is taking place, no queries can be answered, as the worker threads are unable to access the database. Therefore, modifications should be kept to a minimum and should run as fast as possible in order to ensure that this downtime has as little effect as possible. Also, with this scheme, there can only be one modification thread at a time. If there is more than one, these two would need to first be synchronized with each other, to ensure that they do not cause database inconsistencies.

4 Implementation of an Oracle server

This section describes how the first prototype of an intra-AS Oracle server was built. It explains the choice of programming language as well as go into some details of the objects, the rough internal working and their interaction with each other. In the end, the usage of the program will be explained briefly.

4.1 Choice of programming language

The Oracle server is required to process as many requests as possible. For the choice of programming language, this means that an interpreted language is out of the question. Another issue is that the language should be free, open and usable by anyone who wishes to run this server implementation. In the end, C was chosen, as it fulfils all necessary requirements.

However, there have been some difficulties with C when it comes to multitasking, and especially to protecting one thread's memory from another. This language does not offer all the necessary tools to easily use instance variables or any separation of memory. In C, memory is either local or global. There is nothing like an instance variable.

Although the first parts of the server were implemented in pure C, the code was later ported to C++ to have the benefit of encapsulating data within object instances. This also means loss of speed, but as the server was still in its very early stages, it was not possible to say in what way the speed would degrade if a switch from C to C++ is made. Everything that was done in C was ported to C++. The current implementation of the Oracle Server is written purely in C++.

4.2 Development tools

When developing software, it is necessary to have the right tools at hand. The very first and basic thing needed is an IDE which allows for code completion, syntax highlighting, debugging and refactoring. The choice here fell to KDevelop [6] as it has all the requested features, comes in a stable package from the distribution repositories for every major Linux flavour and is able to generate Makefiles as well as the configure command a project needs if it is to compile without the IDE on different machines.

Another important asset when developing software in C/C++ is that the programmer needs to keep track of Memory. Unlike Java, C/C++ leaves the whole memory management to the programmer and does not clean up unused memory autonomously. Therefore, it is possible to lose memory in the program if not every allocated memory block is accounted for.

This calls for correct allocation and disposal of memory. Otherwise, reference to granted memory blocks can be lost, creating a memory leak as the program can no longer find the allocated blocks. To guarantee the lack of memory leaks caused by programming mistakes, Valgrind[15] is used throughout the entire development process.

Valgrind is a free open source project. It overloads the memory allocation and deallocation methods of the operating system for a specific program and keeps track of what was requested, granted and freed. Running a program through Valgrind will tell where memory was allocated which was never freed. This leads to the leak itself and makes it possible to fix. The Oracle server has been run through Valgrind continuously and is known to have no memory leaks in its current implementation.

The downside of using Valgrind is that it considerably degrades the speed of execution. Therefore, Valgrind can be used to test programs but is not intended to be applied in the real use case. Thus every change made to the Oracle server should trigger a test series run of Valgrind to make sure that no memory leaks are introduced from any changes before putting the new version into productive use.

The last tool being used is Callgrind[15]. It comes from the same tool-suite as Valgrind. Callgrind has the same characteristics as Valgrind but monitors CPU cycles spent on operations. Together with KCachegrind[5], a graphical front-end for Callgrind, it is possible to examine closely where a program spends CPU cycles. This allows for speed optimization in the software's most frequented set of code blocks.

KDevelop comes with integration for Valgrind, Callgrind and KCachegrind.

4.3 Data types

The Oracle server needs its own data types to pass information around and to aggregate information which belongs together. The data types originally come from the C version of the Oracle but have not been replaced by objects since they still work in C++ and are faster than object instances passed through the stack during function calls. Data types are used where information needs to be aggregated into a larger block of memory, which no predefined type is able to hold. Also, data types don't implement functions or methods to manipulate them. In the end they are a block of memory with a specific layout.

This list shows the major data types used within the Oracle server:

- oracleHeader
  This is the Header description of a Packet sent to the Oracle server.
- oracleAuthData
  Designed to hold the authentication data of an Oracle packet.
- oracleTLVHeader
  Definition of the TLV header which is used within the payload of the Oracle protocol.
- ipv4Address, ipv4Netmask
  These are unions that allow access to single bytes within the 32 bit int of an IP address or subnet mask.
- routerLink
  Defines the data for a link that exists between two routers.
- ipv4NetworkRouter
  This describes an external network within the Oracle server.
- ipv4NetworkListEntry
  A list entry that is held within the binary tree of the Oracle that describes the external networks.
- routingtable
  The basic entity of which the binary tree consists.
- networkPathProperties
  A structure to hold the aggregated information about a path. This is the main format of the path cache within the Oracle.
- networkConfig
  This can be passed to the constructor of the Oracle server and configures the Oracle server with the given parameters.
- threadParameters
  Holds all information that a thread needs when it is started.

All of these data types can be found in the datatypes.h within the source directory of the Oracle server.

4.4 Constants

A constant in C++ is an expression replaced by the pre-processor by its defined equivalent. This is usually used to simplify the source code or to make it more generic to changes for later times. For example, instead of fixing an invalid IP address as 0.0.0.0 (or 0 if written as an int) in the source code 20 times at different locations, a constant is defined which reads IPV4_NULL_ADDRESS. So, if this address

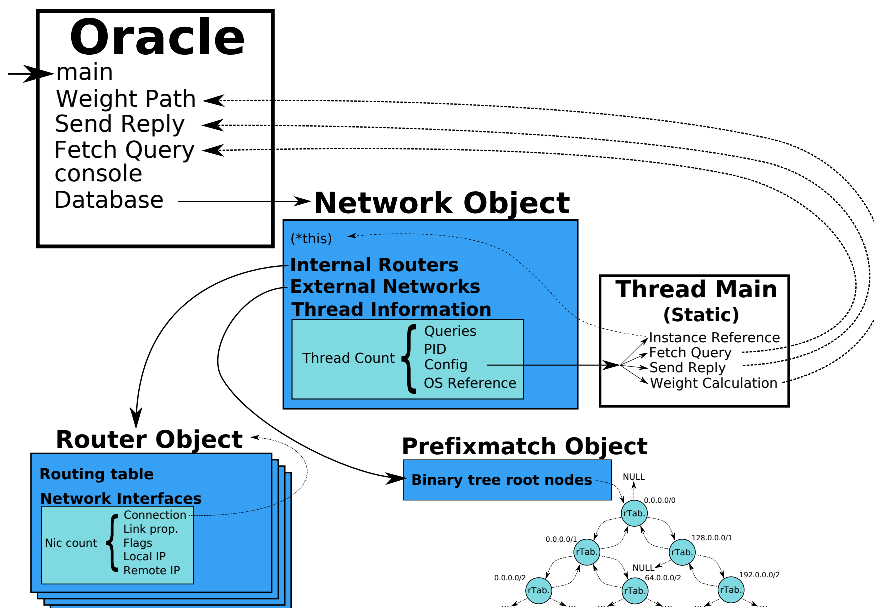

Figure 4: General object layout of the Oracle server

ever changes it only needs to be changed in one place. However, constants cannot change their value, as they are hard coded into the compiled executable. For a change in a constant, the program needs to be recompiled.

The Oracle has two different types of constants. One is used to replace fixed numbers, IPs or parameters to make the code more generic for later stages. The other types of constants are error codes. As it is hard to figure out what error -122 means, all error codes have been replaced by constants. Instead of the return code -122 a function uses the constant RC_INVALID_NIC_COUNT which makes the code immediately understandable to anyone intending to modify it.

Error codes have been used throughout the entire Oracle server to unify error handling and make it possible to give feedback to an administrator using the server, as they are used by the console to supply error messages.

All Oracle-wide constants can be found in the datatypes.h.

4.5 Internal object structure

The Oracle server splits itself into several objects which hold the basic data structures for its operation. Every object is seen as its own instance with its own internal data and is distinguishable from other instances. All objects mentioned here contain internal data as well as methods to change this data.

The general layout of the objects and how they are using each other can be seen in figure 4. The internal structure of the Oracle server is using three different Objects to build its database on the internal network it serves. The Router object handles the routers known to the Oracle, the Prefixmatch holds all information about the external networks and the Network object combines the two into one database capable of supplying network information. The Network object also manages the handling of the threads.

Noticeable is that there are multiple instances of Router objects while only one instance of the Prefixmatch and Network object is sufficient. Each object will be explained in more detail in the following sections.

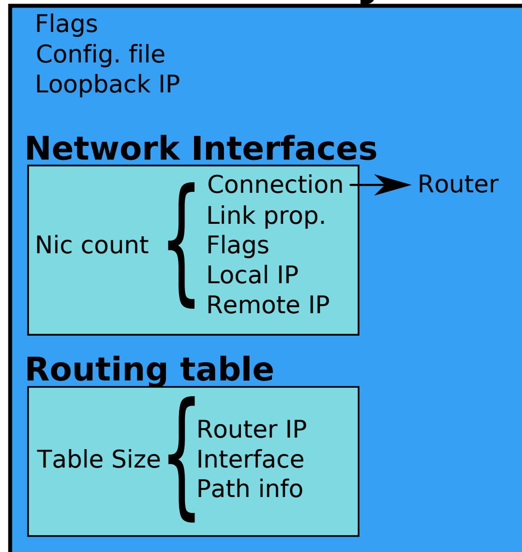

Figure 5: Internal data structure of the router object

4.5.1 Router

The Router object describes exactly one router in the internal network topology of the Oracle. The structure of a Router is described in figure 5. It is the basic entity forming the internal network of the Oracle server database.

As this type of object knows nothing of the network itself, it is not able to connect itself to other routers without the help of a global view. The links defined are asynchronous and specify the outgoing properties. Therefore, a link between two routers can have a different incoming and outgoing characteristic assigned to it.

Each router holds its own internal routing table organized as a hash table. Each entry contains one single IP address of another router together with an outgoing network card. A subnet aggregation is neither desirable nor supported in this case, as the amount of routes the router can hold is fixed by the total amount of routers the Oracle can handle. Along with every route to a router, the aggregated path information is stored for fast lookup of the path. This has to be pre-calculated before it can be used, but once completed, it prevents the Oracle having to traverse the entire network in order to acquire the path information for a specific router.

Every router has its own routing table and therefore its own cache. It is therefore plain to see that in the case of a one-time calculation of all paths, there is no need to use a path-finding algorithm unless the network changes.

However, the routers do not have a global view of the network, and act in their forwarding decision as a normal router would do. Therefore, they are incapable of filling their path cache autonomously, as they administer merely the next hop and not the full path through the network. The path cache needs to be filled by an instance with a global network view.

4.5.2 Prefixmatch

The Prefixmatch object only consists of a binary tree and the necessary modification functions for it. The internal memory structure can be seen in figure 6.

As mentioned before, the Oracle server uses multithreading to boost its performance. However, in

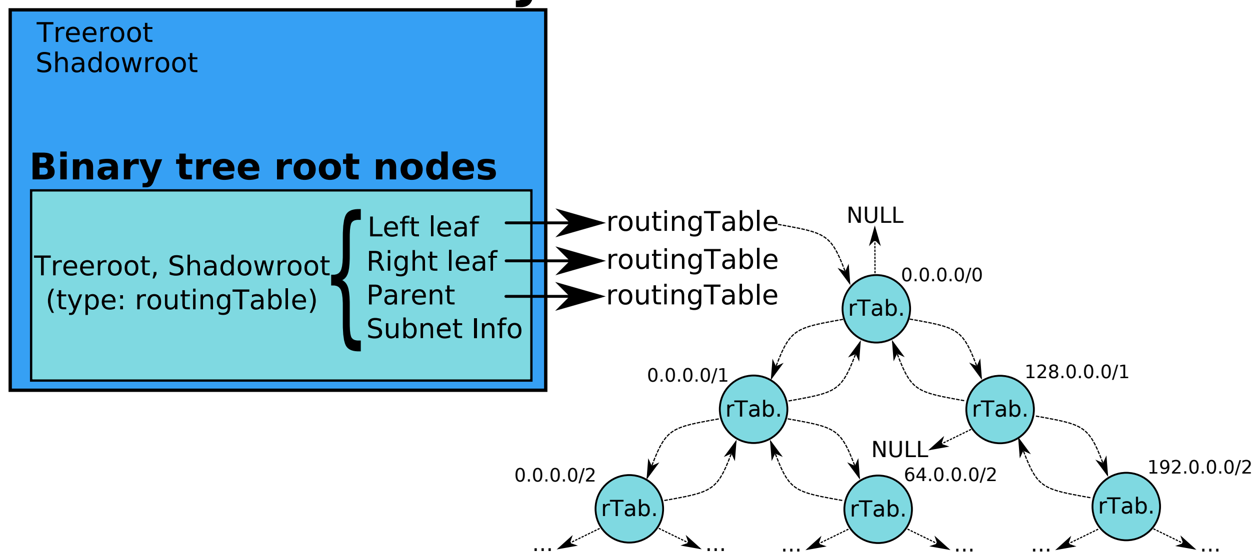

Figure 6: Internal data structure of the Prefixmatch object

doing so, the worker threads can no longer be used for modifications. Also, it would be necessary to stop all worker threads to make sure no thread is reading anything currently being modified in the binary tree.

To cut this downtime to a minimum, Prefixmatch uses a shadow copy of the tree. This allows for modifications on subnet level while keeping the worker threads running on the original. This also makes it possible to do multiple changes to the subnets without having to stop the worker threads each time. Once all changes have been applied to the shadow copy, the worker threads need to be synchronized (i.e. temporarily stopped). A simple swap of two pointers switches the shadow and the original tree. As soon as these are switched, the worker threads can go back to work immediately.

Leaving the worker threads running while switching the entry pointers is not possible, as they can still be reading in the now obsolete copy of the binary tree, as the modification thread starts to delete this memory structure. This creates a race condition between the worker thread finishing and the modification thread deleting the entries. This condition could potentially crash the entire server.

4.5.3 Networklist

This object is a basis stack implementation to pass an unknown number of IP addresses or subnets through the Oracle server. This instance only uses the three basic commands known for a stack: push, top and pop. The internal structure of the object can be seen in figure 7.

4.5.4 Network

The Network object is the database back end for the Oracle server. Currently there is only one database instance used by the server which equals an entire internal network. While Prefixmatch and Router are basic entities for the database, Network uses these two to build its internal representation of the network. To do so, it needs the following information usually supplied via configuration files:

- Configuration files
  The configuration files inform the Network object what global settings are to be used for the server, as well as where the configuration of the routers can be found. Each router can have its own

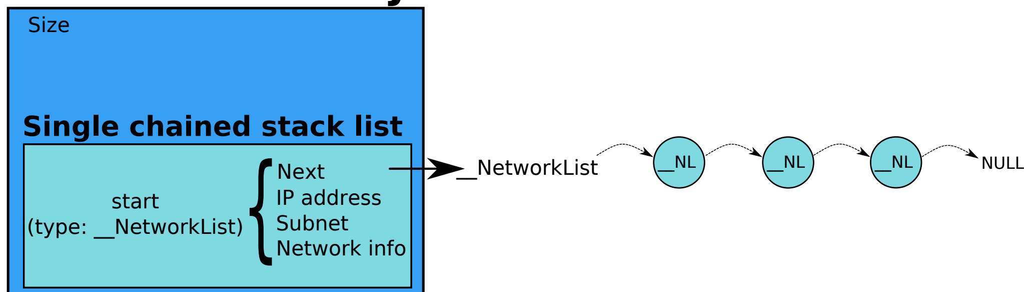

Figure 7: Internal data structure of the Networklist object

configuration file for specific settings. If this file is present, it is passed down to the Router object which then uses it to configure itself.

- Command line options
  Everything except the subnet definitions can be given to the Oracle without the need to specify it in the configuration file. This can be done via the command line options, which are passed as a networkConfig type.

The internal data structure of the Network object is quite large. It holds exactly one Prefixmatch object to save all the external networks as well as multiple router object instances organized in a hash table for fast access. This forms the database back end holding the internal network while applying the necessary abstraction to reduce the memory requirements according to section 3.2.3. Figure 8 shows the memory structure of the Network object.

As mentioned earlier, the routers themselves cannot find routes to subnets, populate their path cache or find paths to other routers, as they are unaware of the global network. The Network object solves this problem. Once the configuration has finished loading, it runs over the network cards of the routers and finds the specified connections. It creates the needed links and sets their respected values the resemble the network.

After the network has been connected, an optional step may be taken. If enabled, the Network object can generate the routing tables for the internal routers based on the assigned weights for the links. This is realized by running a depth first search from every enabled and connected network card of every router based on the assigned link weights. The search continues as long as the total weights for the routes decrease, or are not yet defined.

Thus, every router receives a route to every other reachable router through the path yielding the lowest internal weight. If this feature is disabled, the routes will be loaded from the configuration files and are not changed. This option was specifically intended to be an optional feature as there may be cases where an administrator opts to set the routes manually.

The last step the Network object takes before it becomes fully operational is to fill the path cache. If the routing tables are being calculated, the path cache is already filled from the previous calculation. If the routes are being set manually, the Network object will fill the path cache to make sure that every specified path is pre-cached and easily accessible. Figure 9 shows the load process as a state machine.

Once the Database back end has been fully loaded, the threads need to be initiated to serve requests from clients. While the worker threads all run in the Network object, they are not a part of the instance. This is mainly due to the fact that the chosen thread model creating the POSIX[8] threads is purely C based and does not support C++ object references. As C is unaware of any object reference, it is not possible to pass the **this* pointer of the Network object directly to the spawned thread. The **this* pointer of an object tells the code which instance of the object the program is currently running in. If it

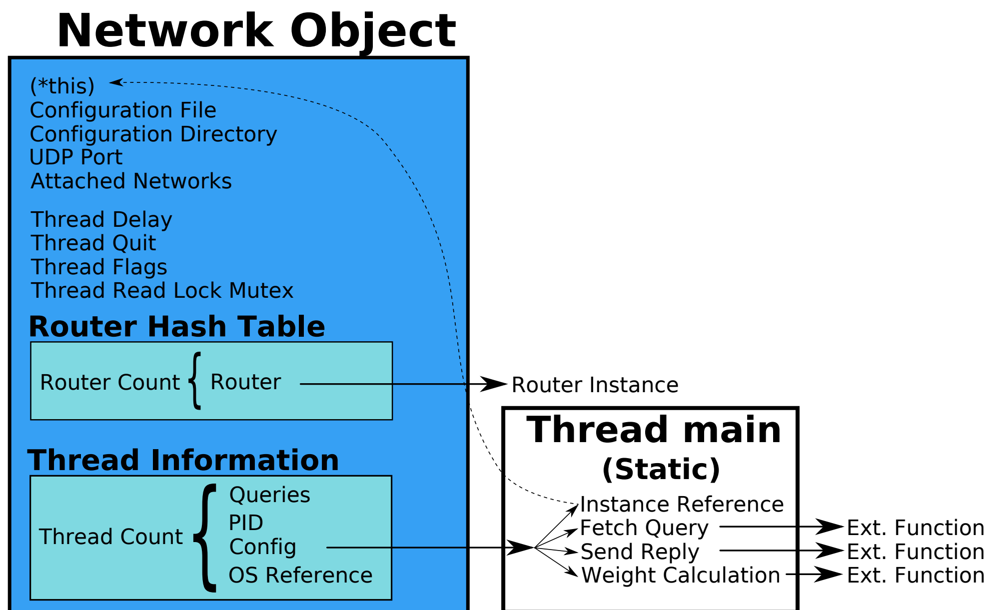

Figure 8: Internal data structure of the Network object

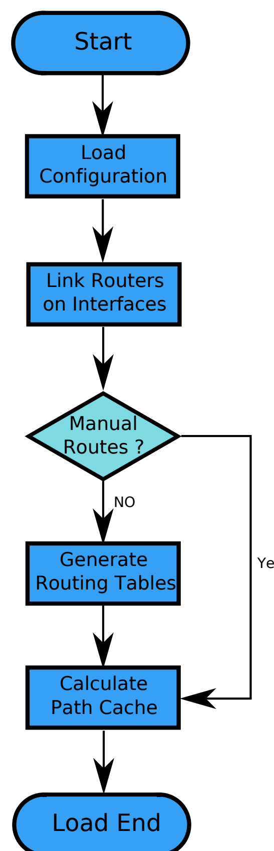

Figure 9: Network object load process

is missing, the starting thread does not know what Network instance it is spawned from and cannot call any functions or access any of its memory.

Along with this inability to pass the **this* pointer to the thread comes the problem that, since the **this* is missing, no dynamic function of the instance can be used as an entry point for the thread, as the calling function knows nothing of the object instance and thus cannot find the appropriate entry point. Therefore, the call to start the thread must be done to a static function of the object outside the instance not needing a **this* pointer.

This makes it impossible for a thread to access the database. A work around this is to pass the **this* reference from the thread creating function in the instance as an argument and extract it on the other side when the thread has started running in the static member. This enables the thread to find out which instance called it, and have a hook back into the database. This also makes it awkward to work with, as the **this* pointer is not automatically resolved but needs to be referenced manually.

Along with the **this* reference of the calling instance, there are three callback pointers for functions passed to the thread as seen on the right in figure 8 . The first function is to receive requests from clients. As the socket handling and packet decoding can become quite large, this must be separated entirely from the Network object to make later modification in the Proxidor protocol easier to incorporate. It also allows the possibility of having different functions plugged in at runtime after the code has been compiled. The reason this can be done is that the receiving function does not need any database access.

The second argument which is passed down is the send function that is needed to deliver a reply to a request. It is a callback function for the same reason as the receive.

The third function that is passed is the weight calculation function. This determines the actual weight or expense a connection from a source to a destination has. As these metrics can vary for many different applications of the Oracle, it is crucial that this can be easily found and modified in order to ensure that an administrator can implement their own weighting system. The function is kept simple, as it receives the entire aggregated path information coming from the database and only expects a single floating point in return, representing the expense of the path then used to sort the list of IPs in the request.

4.5.5 Oracle

Oracle is not an object, but a set of functions. These include parsing of command line arguments, the socket handling as well as the weighting function. In addition to this, the command line can be found here. As all worker threads are started through the Network object, the modification thread is required as well to make changes to the database while it is running. The modification thread is actually not a thread but the main program itself after all the worker threads are spawned off.

Figure 4 shows how the Network object links the threads to the general weighting function. The reason behind this callback function is to supply an easy method of modifying the ranking the Oracle performs without having to go through the source code. The ranking gets handed a NetworkPathProperties Datatype and returns a single floating point value. Each IP is assigned a weight, making it possible to decide on the quality of destinations. The lower the value for a destination, the earlier it appears in the ranking.

The current ranking function working the Oracle can be seen in equation 3. It is derived from the idea that hops and especially AS borders should be punished. Weighting the AS 16 times as high as an internal hop will keep traffic local. Destinations found on the same Network or in a neighbouring AS will always end up at the top of the ranking. The bandwidth is added in case two possible destinations have the same weight for the distance. In this case the destination with the higher available bandwidth will be ranked better.

$$weight = 8 * AS_distance + 0.5 * internal_hops + \frac{10}{minimal_bandwidth} \tag{3}$$

This ranking function is used as a test sample and does not have any relation to a real world example. As previously discussed, each ISP wishing to utilize the Oracle in their network can implement their own ranking function by simply changing this calculation. The full metrics passed to calculate the weight are

- minimal bandwidth
- aggregated latency
- total hops
- AS number
- AS distance
- total internal weight
- (defined congestion)

This list can be extended to pass more information into the ranking function. Possible additional fields are first and last hop bandwidth, individual penalty weights for peering links to other AS and distance in the real world. Also, the congestion is already defined in the metrics but is currently not being distributed in the Oracle server and will always be zero in its current state. This is due to the fact that congestion is very hard to measure and cannot be estimated on a general basis.

4.6 Configuration files

There are two different types of configuration files for the Oracle server. One is the main- or global configuration of the Oracle server itself. It holds the maximum amount of routers, how many threads should be started to serve queries, how the external networks are configured and where the configuration directory for the routers can be found. A sample of a configuration file for the Oracle server can be found in figure 10.

The other type of configuration file is a router configuration. Although not every router needs a configuration file, they are not part of the network without these files, as they stay unconnected and disabled in their default setup.

The configuration file of a router specifies the setup of the network cards a router has, the link properties applying for a connection and what status the router should be in after the initial setup. A sample of a router configuration file can be found in figure 11.

The console of the Oracle makes it possible to dump the current working configuration of the entire Oracle into text files. Therefore, it is possible to start the Oracle without a configuration, configure it and dump the created setup to the files.

4.7 The console

The console of the Oracle was not originally intended to be a part of the first version. But as interaction with the server was eventually required, the need for a console arose.

The current version of the Oracle includes a command line tool which the server starts as soon as the worker threads have been spawned. Currently there is no way to connect to the console via a remote port, but a feature like this may be added sometime in the future.

The console offers commands to change, reload or reset various aspects of the server. If an empty line is submitted to the server, it will print out the basic help for commands. Also, if a command is incomplete, the help for this command is displayed via the console to give administrators a quick overview of the command syntax.

Upon successfully submitting a command, the console will always display a response from the server. The most common answer will be a simple "operation successful" but it is also possible to trigger errors from the server.

As previously mentioned, all errors from the server are replaced by constants. Likewise, all error constants are translated into an understandable, simple message and not some arbitrary number, dispensing with the need to look up a suitable explanation. Figure 12 shows an example of the command console.

```
configdir /etc/oracle/config/
routers 40
threads 2

router 10.0.0.1 15
router 10.0.0.2 15
router 10.0.0.3 15
router 10.0.0.4 15
router 10.8.0.1 15
router 10.8.0.2 15

snetwork 195.37.16.240/28 10.0.0.2 14
snetwork 195.37.16.224/28 10.0.0.1 14
snetwork 127.0.0.1/32 10.0.0.4 14
snetwork 192.168.0.0/16 10.0.0.4 14

enetwork 0.0.0.0/2 10.8.0.1 14 1 5
enetwork 0.0.0.0/4 10.8.0.2 14 2778 6
```

Figure 10: Sample configuration file for the Oracle server

```
nic 0 10.0.1.4 10.0.1.3 UP
nic 1 10.0.1.6 10.0.1.5 UP
nic 2 10.0.1.14 10.0.1.13 UP
nic 5 10.0.1.12 10.0.1.11 UP

link 0 35 100 0 1
link 1 35 10 0 1
link 2 40 10 0 1
link 5 4 10 0 1

route 0 10.0.0.1
route 1 10.0.0.2
route 5 10.0.0.3
route 4 10.8.0.1
route 4 10.8.0.2
```

Figure 11: Sample configuration file for a router within the Oracle server

```
Oracle :>
basic commands in Oracle
  print
  set
  reload
  dump
  thread
  create
  delete

 type one of the commands to get more specific help
Oracle :> print networks SRC
[SOURCE NETWORK] 127.0.0.1/32
  Router router 10.0.0.4 nic 14
[SOURCE NETWORK] 192.168.0.0/16
  Router router 10.0.0.4 nic 14
[SOURCE NETWORK] 195.37.16.224/28
  Router router 10.0.0.1 nic 14
[SOURCE NETWORK] 195.37.16.240/28
  Router router 10.0.0.2 nic 14
Oracle :>
```

Figure 12: Screen shot of the oracle command line console

4.8 Socket handling

For the Oracle server to be capable of handling requests from the Internet, a socket through which the server listens is required. At the moment, the standard Linux AF_INET socket is used [9]. Currently, the Oracle uses the UDP protocol on port 8899, but this will most likely change in the future, as this port number is already defined.

The reason the server is using UDP as its communication protocol is that UDP is faster and uses fewer resources than TCP. With UDP, there is only one packet that needs to travel to the server and one that needs to go back. This simple answer-query scheme eradicates the need for connection handling. In the event of a packet being lost on the way, the request will simply be sent again.

One thing that deserves some investigation is the byte order in the packet. This poses no problem for communication between machines with the same architecture. However, there is a problem when the packets are sent over the Internet to machines of unknown architecture. To circumvent this, the packet generation and parsing functions always assume everything in the packet larger than one byte to be in network byte order.

This implementation of the Oracle server is currently only limited to IPv4 addresses. The Internet is currently slowly shifting towards IPv6 as the address spaces for IPv4 networks are running out. This means that in the future there is a possibility for the protocol to transport sorted lists of mixed IPv4, IPv6 and possibly subnet declarations in one single packet. The best solution for this at the moment is the Proxidor[2] protocol which is currently under development. It is a highly modular protocol which can carry any payload due to its custom payload headers.

4.8.1 The proxidor protocol headers

The Proxidor protocol is currently being developed within the ALTO research group. Proxidor stands for Proximity-Identificator-Oracle. The Oracle server is using version 0.2 of this protocol to handle its communication between the client and the server.

The protocol is split into multiple parts. These parts are the main header, the main header extension

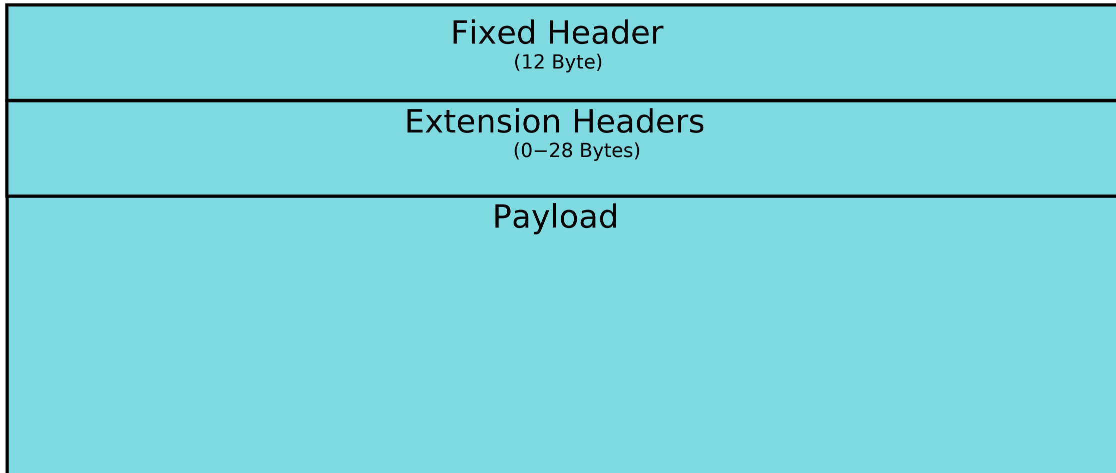

Figure 13: Proxidor Protocol v0.2 General Header definition

and the payload being carried. The general query layout can be seen in figure 13. As one of the requirements of future Oracle implementations may be to support more than just one type of IP address, there must be a way to carry different types of IP addresses in a sorted order in the packet. To achieve this, the payload itself is again split into several parts, each composed of a header, an option extension and the real data.

Figure 14 shows the basic layout of the header of any packet the server is sending or receiving. As there is no difference between the layout of a request or a response, this header is used universally throughout the Oracle as well as the client. This is still an experimental design of the protocol implementation and does not use all fields specified. It only utilizes those it requires for the basic communication. Used read-only fields are marked as checked in the figure; modified fields are labelled modified, while unused fields are left blank. The following list explains briefly every field's intended use, even if it is not used by the Oracle.

- **Version**
  This, in later versions of the Proxidor protocol, will explain what version this packet currently is. This field is only checked to be zero.

- **Servcode**
  This resembles a field set of single bit flags. This is either used by the server to tell a client which metrics the server supports, or by the client to request a certain metric from the server. In the current implementation, there is only one metric available. This field is only checked to be zero by the Oracle.

- **Flags**
  The flags are used by the client and the server to give information about the packet. The exact bit pattern is shown in figure 15. Table 1 lists the usage of the Flags.

- **Identification**
  This 32-Bit field is used to identify a reply to a certain request by the client. This has nothing to do with the server and is not touched by it but returned as-is to the client.

- **TTL**
  The time to live (TTL) indicates how long the server considers the given response as valid. This is to inform a client how long it can cache the replies from the server before they become stale. This

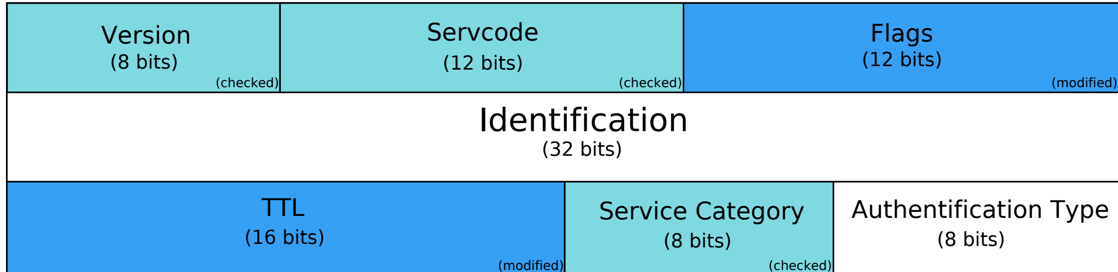

Figure 14: Proxidor Protocol v0.2 Fixed Header definition

should only be set by the server, but the server does not check it, only currently overwriting it with a static number of 60.

- **Service Category**
  This enables the Proxidor protocol to have different types of payloads attached to it. Dependent on the service category is how the parsing of the packet should continue. The service category for an Oracle request is zero. Therefore, the field has to be set to zero as well.

- **Authentication Type**
  This determines the type of authentication that is used in the authentication header. The current implementation neither supports nor checks this field as any authentication is currently skipped.

If the AD flag is set to request, the server knows that there will be a 64 Bit (eight byte) block of authentication data directly following the fixed header. In the current implementation, the server is able to handle the existence of this information, but will skip it if it is supplied. Authentication itself is not supported yet.

After the fixed header and authentication data follows the payload of the packet. As seen in the flags the only service category the server can handle at the moment are requests. The payloads for Oracle queries and responses are organised in TLVs. TLV stands for Type-Length-Value and defines a sub-header as well as a sub-payload contained in the packet. The layout of a TLV header and its payload can be seen in figure 16 while the flags and fields of the TLV header are described in table 2.

The current Oracle implementation is capable of handling multiple TLVs in one packet, adding them into one big list. It always returns only one TLV containing the entire list, even if it was sent as multiple TLVs. It can also figure out the size of the attributes and skip them accordingly, but does not include any attributes in its replies. Also, it always deletes all attributes which were sent to it via a request.

The definition of these headers makes it possible to calculate the maximum number of IP addresses being carried in one packet. The Oracle is using UDP as its transport protocol. The maximum size of a UDP packet is 1500 including all header and management information. Subtracting the fixed header (12 bytes), the Authentification data (8 bytes) and one TLV (4 bytes) leaves 1476 Bytes for storing IP addresses.

The limit is dropped further due to the protocol already defining an extension to supply source IP addresses and subnets. An IPv4 address uses four bytes, while an IPv6 address uses 16 Bytes. Together with a subnet definition of one byte each, this means using another 22 bytes for the sources. Using the already reduced payload size of 1476 Bytes and taking these additional definitions into account leaves 1454 bytes to be spent on the payload. This makes a total of 363 possible IPv4 addresses candidates being sent in one request. In the case of IPv6 this supplies space for 90 IPs per packet.

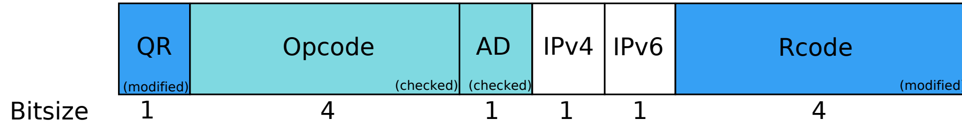

Figure 15: Proxidor Protocol v0.2 Fixed Header Flag definition

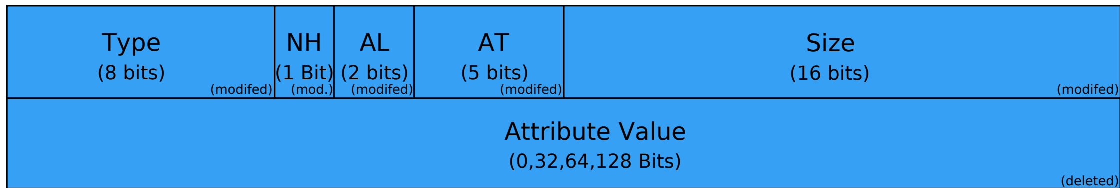

Figure 16: Proxidor Protocol v0.2 TLV Header definition

Flag name	Use
QR	Tells is this packet is a request or a response (0 = request, 1= reply)
Opcode	Defines type of the request. Currently only 0 is used as COQ (Client Oracle Query)
AD	Flag to tell if authentication Data is attached to this header (0=false, 1=true)
IPv4	IPv4 source address (unused at the moment)
IPv6	IPv4 source address (unused at the moment)
Rcode	Rcode tells the type of the reply. Currently only 0 is used to indicate "No Error"

Table 1: Proxidor Protocol v0.2 Fixed Header Flag description

Flag name	Use
Type	Defines the TLVs payload. Currently Type 1 is support (IPv4 IP List)
NH	Indication of another TLV following (0=false, 1=true)
AL	Length of the Attribute Value field (0=disabled, 1= 32 Bit, 2=64 Bit, 3=128 Bit)
AT	Type of the attribute that is attached (unused at the moment)
Size	Size of the TLV including the Header, attributes and the Payload
Attribute Value	Value of the TLVs attribute (currently unused)

Table 2: Proxidor Protocol v0.2 TLV Header Flag description

4.9 Additional server modes

The Oracle server has no implementation apart from the prototype implemented during this thesis. Hence there are currently no clients on the Internet to use the Oracle's service, meaning there is no real-life environment to test the server. To build a test setup, a client for querying the server was integrated into the developed software.

The Oracle server currently supports the run modes "server" and "client". While running in server mode, the Oracle will behave as was described throughout this chapter. It will load the database provided by the configuration files and serve requests from its known network.

4.9.1 Query client mode

The client mode was built into the server to have the ability to test the setup of an Oracle. It was originally intended to be a very small and simple implementation for querying, but was later enhanced to have a more complex structure as the test cases became more sophisticated.

The client supports the following modes:

1. **ping**
   In its simple form, the client will act the same as the normal ping utility provided with any modern operating system. Instead of using ICMP to "ping" a machine this will send random queries to the specified server and wait for an answer. The output of this mode is matched closely to resemble the ping utility.

2. **multi-client**
   For performance tests, one single client is not enough to query the server. Therefore, the client is able to spawn multiple threads in multi-client mode to query the server. This also enables the client to synchronize the various threads. When it comes to performance tests, it is required that all clients start querying simultaneously. The client can do this via the synchronized start option. In this mode, all client threads are spawned but are not allowed to start querying until the spawning is complete.
   Likewise, there is a synchronized stop which can be enabled. If the clients are set to send a specific amount of queries, this option will terminate all threads once one reaches the given number of requests. This will avoid slower threads speeding up towards the end when others have completed their workload.
   The output of this mode is close to the normal output, but lists the results in a table. Aggregated statistics are supplied on a per thread basis. This mode is mainly used to determine the server's performance.

3. **flooding**
   In flooding mode, the client no longer waits for replies but pushes requests out as rapidly as possible. It also does not allow the use of multiple threads, being that this scheme will overload any server, supposing it is supported by the bandwidth. The statistical output of the flooding client is reduced as it provides no information on how many requests were answered or how much time was taken.

Having these three modes available makes it possible to set up test environment and generate results about the behaviour and performance of this implementation.

```
The Oracle Server
 required options
  -mode [server|client]  mode to start in

 server mode options
  -config configfile     config file to start on [default ./oracle.conf]
                          The config file will OVERWRITE any commandline options if existant there!
  -configdir dir         Path to the configuration directory of the routers [default .]
  -maxrouter             maximum number of routers the oracle can handle [default 100]
  -threads no            Number of workerThreads (max. 32) [default 1]
  -threaddelay ms        milliseconds to wait between requests [default 0]
  -iroutes [expl|impl]   handling of internal routes (explicit or implicit) [default explicit]
  -port portNO           UDP port of the Server [default 8899]

 General client mode options
   -server IP            IP address of server [default: 127.0.0.1]
   -port portNO          UDP port of the Server [default 8899]
   -delay ms             Delay between queries [default: disabled]
   -count no             max. no of queries to send [default: unlimimted]
   -size count           number of ip's per query [default: 100]
   -timeout ms           timeout in milliseconds to wait for a reply [default: 500]

  Single-client mode options
   -verbose              makes the client print out mode information
   -flood                sends out requests without waiting for replies
   -clockdelay n         This will wait n processor instructions [experimental]
   -persec n             sends approx. n packets per second [experimental]

  Multi-client mode options
   -threads              number of clients to spawn [default: 1]
   -syncstart            all threads will start at the exact same time
   -syncstop             if one thread is done all will terminate at that point
```

Figure 17: Oracle commandline options

5 Server performance test setup

This chapter explains the experimental setup of the Oracle server implementation from section 4. It will go into the hardware used, the setup of the software and the composition of the test series.

5.1 Test setup

The test setup for the Oracle embodies two servers. The hardware configuration of the machines can be seen in table 5.1. They are switched via a 1 Gbit link. There are no other hardware components attached to this network, thus there is no traffic being generated in the network to affect the test results.

The Oracle test client is used to send queries to an Oracle server instance. Since the load for generating packets is far lower than the ranking the server must do, it is possible to have only one machine that runs multiple clients. Due to the two quad-core CPUs of the hardware, the Oracle is running with up to eight real CPUs at its disposal.

5.2 Test network

To simulate an internal network, the Oracle is serving requests for the network shown in figure 18. This does not match the real topology the Oracle was run in, as the test setup only has two computers. Because of that, all requests were from the same source, meaning all queries came from the same source router. The destinations in the queries were randomly generated.

This test network consists of 216 routers spread over nine network hubs connected via high performance connections.

Each network hub consists of twelve clients, ten intermediate and two backbone/Uplink routers interconnecting the different network hubs. The solid lines in the figure represent primary links. The internal weight for path calculation is set to one on these links. The dotted lines are backup connections. They have less bandwidth defined and a higher latency assigned to them. Their internal weight is set to two. The routing tables of the routers are calculated automatically via the weights assigned to the links.

	Server	Client
processor	Intel(R) Xeon(R) CPU L5420	Intel(R) Xeon(R) CPU L5420
CPUs	2	2
cores per CPU	4	4
Mhz	2500 MHz	2500 MHz
cache	6144 Kb	6144 Kb
RAM	16 Gb	16 Gb
Operating System	Debian Etch	Debian Etch
Kernel	2.6.18-6-amd64	2.6.18-6-xen-amd64
Network cards	Intel 82571EB	Intel 82571EB

Table 3: Oracle test hardware setup

Each router in a hub network has at least two independent links to the backbone/uplink routers. The backbone routers themselves are organized as a binary tree with one primary and one secondary line running from one network hub to the next. Routers marked with a black symbol are exit points to the rest of the Internet via other autonomous systems (AS). An AS is a network that is under the control of one administration. An ISP is usually considered to be an AS, but can consist of more than one.

In this network, there are three exit points which hold about 67,000 routes to other networks. These routes were randomly generated and dumped into the configuration of the server. Although the external networks are random, they remain the same every time the server loads.

Each router has a loop back IP assigned to it. These are chosen by their position in the network. Since the IPs of the routers have no effect on the path finding, they can be chosen freely with no regard to overloading subnet declarations. Even in the case of a collision between a loop back IP of a router and a subnet that holds the same IP, the Oracle will always prefer the network and never send the connections towards a router. This behaviour comes directly from the layer of abstraction that was done in section 3.2.3. Therefore the IPs for the routers can be chosen arbitrarily.

All routers in the network in figure 18 have an address from 10.0.0.0/8 subnet. The second number is the hub network the router can be found in, the third is relative to the ring the router is attached to. The rings count from the outside to the inside. The last number is to reference the router in the ring. Each ring starts at the left top-most router available and runs counter clockwise. Example IPs are also shown in figure 18.

The clients in the tests that were run are all attached to the router (10.1.0.1) which is located in the top left of the network. It is marked as the source router in figure 18. Also, each client router has one 130.149.x.0/24 attached to it, which makes a total of 108 intra-AS client networks that are distributed evenly throughout the entire internal network. The source network from which all queries to the Oracle originate has the range 130.149.221.0/24 and is a second internal network attached to the query source router.

5.3 Test composition

The Oracle server was run through two different sets of tests with varying parameters. The first test series is concerning the Oracle server's performance, i.e. average response times, maximum number of clients and replies the server is capable of handling with different settings.

The second part of the test looks at denial of service attacks. In this scenario, the Oracle is overloaded with requests of different types of packets.

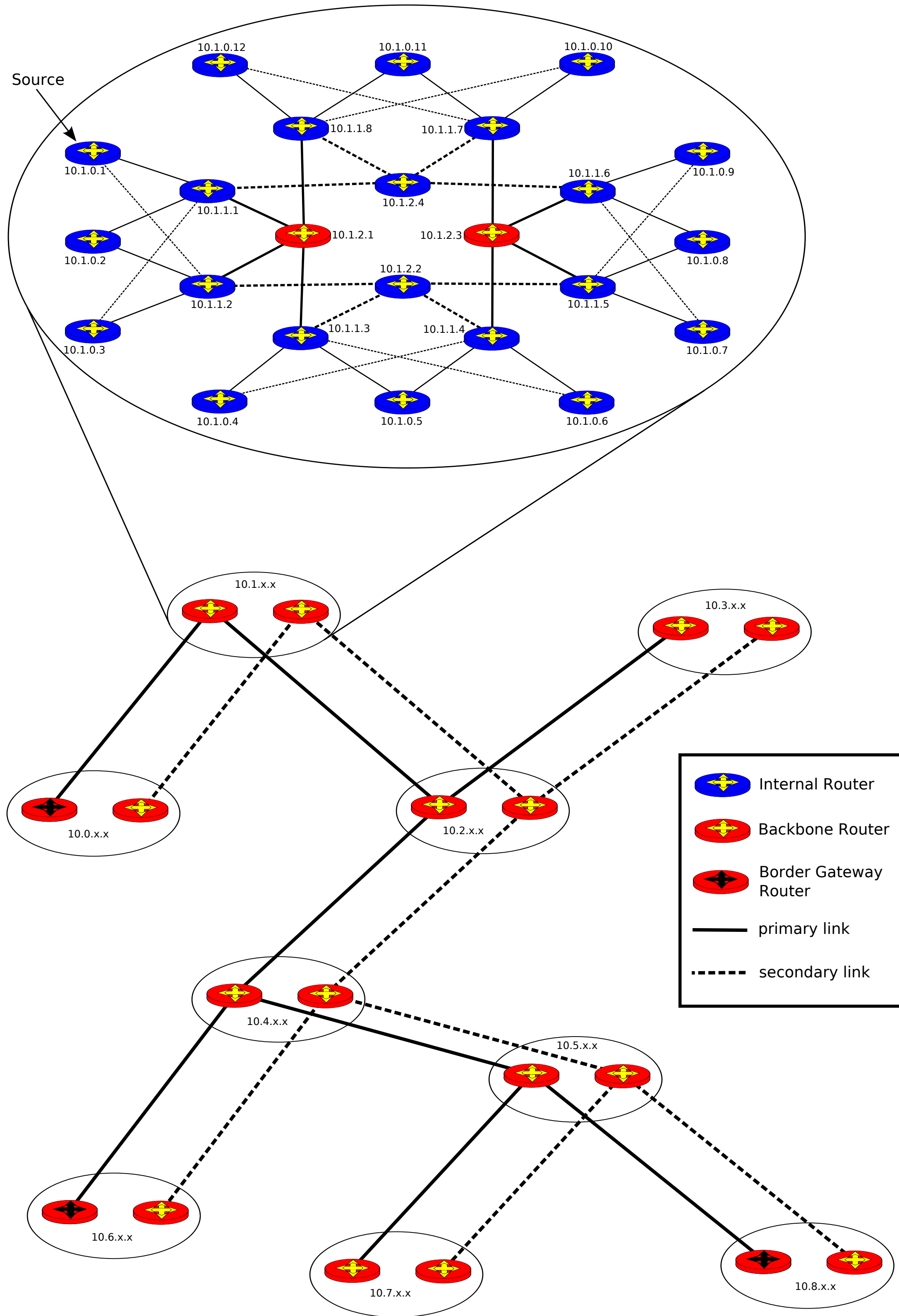

Figure 18: Network loaded into the Oracle during testing

5.3.1 Performace Test Composition

In this setup, the server is to be brought to its maximum performance, while not overloading to the extent of a denial of service attack. This is done via regulating the amount of queries sent to the Oracle by using the packet loss to estimate the rate of querying. These types of clients are called regulated clients.

In client mode, the Oracle spawns 750 threads on the client hardware which are synchronized, meaning that they start and stop unanimously. Each Client is supposed to query the server 2,500 times. The synchronized stop exits all clients as soon as one client has finished its defined number or queries. This is done so that clients experiencing timeouts will not speed up towards the end, when the first threads have finished their tasked workload.

The clients are set with a delay between a reply and the next request. Also there is a timeout set, where a client has to wait before it considers a request lost and sends a new query. The timeout for a reply was set to 50 milliseconds.

The following list discusses the different metrics measured during the tests.

- **delay between requests**
  The delay between the requests determines how many queries per second can be sent to the Oracle in the best case. The delay is always applied, even if there is a timeout. Thus, a query that receives a reply will have a total time between requests of $rtt + delay$ ms while a timed out packet will have a constant delay of $timeout + delay$ ms. Since an Oracle that is overloaded will drop more packets than an idle one, the timeout will regulate how many queries per second are really dispatched.
  The minimum amount of packets the clients will send if all requests are lost is defined by equation 4 . The maximum can be estimated by equation 5.

$$min_requests_per_second = clients * \frac{1000ms}{timeout + delay} \tag{4}$$

$$max_requests_per_second = clients * \frac{1000ms}{min_rtt + delay} \tag{5}$$

  The delay was varied with the values 5, 10, 15, 20, 25, 35 and 50 milliseconds during the testing. This makes a maximum query rate of about 75,000 queries per second if the server needs 5 milliseconds to reply to a request. The less time the Oracle takes to answer the queries, the more requests will be sent to it. On the other side, the minimum query rate the clients will take is about 7,500 per second.
  During the delay tests, every packet sent to the server had 100 random IPs as a payload within it.

- **IPs per request**
  The performance of the server is dependent on the number of IPs per request. This means that the more IPs are in one packet, the longer the calculation of the response should take. This will affect the overall response time of the server as well as the amount of requests per second it can successfully answer.
  The server was queried with 10, 50, 100, 200, 300 and 363 IPs per request. The delay during these tests was fixed on 5 ms per client to always ensure a maximum load on the Oracle.

- **Threads used by the server**
  The third parameter varied was the number of server threads. As multithreading never scales linearly, it is interesting to find out what performance boosts the server gets when the previous tests are re-run against the server with a different amount of threads at its disposal.
  The threads the server was allowed are 2, 4, 6, 8, 16, 24 on a dual quad-core machine. This simulates machines that have less CPUs as well as a 3:1 ratio of threads to CPU. It is obvious that the performance will not increase once there are more threads than CPUs, but it is also important to know that the performance does not decrease when this is the case.

Threads	Delay	IP's per request
2	5	10,50,100,200,300,363
2	5, 10, 15, 20, 25, 35, 50	100
4	5	10, 50, 100 ,200, 300, 363
4	5, 10, 15, 20, 25, 35, 50	100
6	5	10, 50, 100, 200, 300, 363
6	5, 10, 15, 20, 25, 35, 50	100
8	5	10, 50, 100, 200, 300, 363
8	5, 10, 15, 20, 25, 35, 50	100
16	5	10, 50, 100, 200, 300, 363
16	5, 10, 15, 20, 25, 35, 50	100
24	5	10, 50, 100, 200, 300, 363
24	5, 10, 15, 20, 25, 35, 50	100

Table 4: setting of parameters during the performance test of the Oracle server

Table 4 summarizes the settings used during the testing of the Oracle. Every Simulation was run 100 times to ensure that the results are accurate.

5.3.2 Denial of service tests

Denial of service (also known as DoS) attacks on a server basically swamp it with requests, overloading it to the point of it becoming impossible for the server to answer a sufficient amount of requests. The same test setup was used as before, but instead of using 750 threads all acting as their own client, this test was run with just one client sending out requests without waiting for a response. As there are no delays defined in this test, this parameter can be safely dropped from the series.

The chosen number of Oracle server threads during this test series was eight, meaning that the system time needed to handle the software interrupts to pass the packets from the network card to the user space application, is also taken away from the Oracle server.

The number of IPs per request was varied as was done in the tests before, but the lowest possible amount of IPs the server accepts was added to the test series. Therefore, the numbers of IPs per packet were 2, 10, 50, 100, 200, 300, 363. Each time 500 million requests of the specified size were sent as fast as possible.

Since the client does not wait for a response tcpdump [14] was used to count the replies that came back from the server. This means that the entire output of tcpdump was redirected to /dev/null and only counters were used to determine how many answers were received. Each test was re-run 100 times to ensure that the results are accurate.

5.4 Additional Tests

The requirements 1 and 2 from 3.1 state that the server's performance is not dependent on the size of the network it serves. To show this, a very simple network consisting of only ten routers was loaded into the eight thread server and the full test setting from section 5.3.1 was run against it. This consists of running the delay as well as the packet size varying tests on it. The network that was loaded into the Oracle can be seen in Figure 19. The same external prefixes were used for the three external routers. The client Networks were all attached to the two routers at the top while the queries came from a client that was attached to the central router.

The thickness of the lines in this Figure shows how much bandwidth a link between routers has as well as its latency. The thicker the line, the more bandwidth it has and the less latency was assigned to

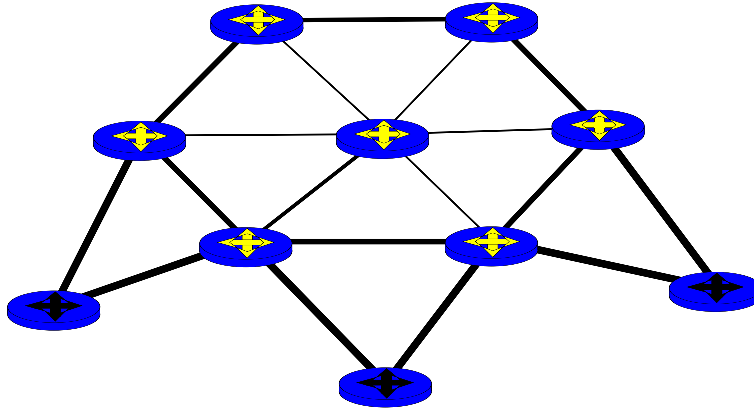

Figure 19: Small internal network for testing

it.

The last test setting that was done was to show the recovery of the server from a denial of service attack as well as how the regulated clients behave.

For this setup, two more machines were needed. The server and the client stayed as they were set up in 5.3.1. The parameters for the client were set to a 5 millisecond delay and 100 IPs per request. In addition to that, a third computer with the same hardware configuration was attached to the network which was to run the DoS attack against the Oracle while the regulated clients were querying it.

The switch between the clients and server was set up to mirror all packets going from and to the server to a fourth computer acting as the log machine. This is needed because the overhead of capturing packets of a full denial of service attack would pull too many resources away if it was run on the client machines or the server.

Of the captured packets, only the first 68 bytes were saved, as it is only interesting from where to where the packets were flowing. This test was only run once and is not reliable in its measured results for how many responses the server can handle under what load, but it does show if and how the server recovers from a denial of service attack as well as if the regulated clients really work.

6 Test results

This section shows how the implementation from section 4 of the Oracle server performs in the tests discussed in section 5. It will present various results of the tests run against the implementation and discuss them.

6.1 Regulated clients

In section 5.3.1 it was claimed that when using a delay and timeout between requests, the amount of queries sent by the client to the Oracle server will regulate itself. In detail, this implies that by an increase in packet loss, the clients will send fewer requests to not swamp the server, yet will always generate a load above that which the server can handle. As this setup is being extensively used throughout nearly all of the test scenarios, it is important to show that this behaviour is really happening. Figure 20 shows the behavior of the regulated clients when the network is suddenly swamped with requests from a DoS attacker. The server was run with eight threads in this scenario. There were 750 regulated clients all querying at what they assumed was the maximum the server could handle, while the DoS client was sending out requests as fast as possible, regardless of what the Oracle was capable of processing. Both types of clients used a payload of 100 IP addresses per request. The bandwidth with which the DoS client pushed out requests was approximated at 80 megabytes per second.

There are two things notable about the behaviour of the client in this test. For one, once the DoS attack on the Oracle server begins, the regulated clients reduce their sending speed to their minimum. The second point noticed is that the regulated clients recover within one second to their previous rate of querying once the DoS attack stops. One more thing point out is that the clients where not run with the synchronized start or stop option. Figure 21 shows results from the same test, but displays the captured packets that the server sent. It can also clearly be seen that once the DoS attack starts, the regulated clients receive barely any answers from the server anymore, while most of the replies go back to the DoS client. However, it can also be observed that due to the heavy overload the server is experiencing, the rate of answering drops to about one third of the normal rate. This is caused by congestion on the link and the software interrupt of the operating system trying to handle the network load. While the congestion increases the packet loss, the software interrupt takes processing power from the Oracle. Also, the answers fluctuate more during the DoS attack than they do compared to the time the server is being queried by the regulated clients. It can also be seen that once the attack stops the server recovers within one second to its previous rate of reply performance.

This test was only run once to produce the required graphs. Therefore no assumptions about the server can be made from this graph. However it is a good indication of how the server will react to a complete overload from one or more sources.

6.2 Replies per second

One of the goals of this implementation of the Oracle server was that its performance be good enough to simultaneously handle multiple clients as well as multiple threads. Figure 22 shows the results from querying the server running with different threads assigned to it.

The plot shows the number of threads the server was allowed to use in comparison to the packets per second. As one packet is the same as one query or response, these can be seen as equal. Each packet consisted of the Proxidor standard and extension headers defined in 4.8.1 carrying a payload of 100 IP addresses. The Oracle was configured to know every IP address on the Internet, therefore the answers from the server have the same size as the queries.

The first remarkable thing about figure 22 is that the number of replies per second increase with the number of Oracle server threads. What can also be seen is that the avoidance of a full overload of the server was accomplished, as the queries per second grew with the answers per second. Yet there is always some packet loss, indicating that the Oracle was not able to handle all requests. The loss cannot be counted towards congestion, as the test setup has a switched connection of 1 GBit/s. The physical

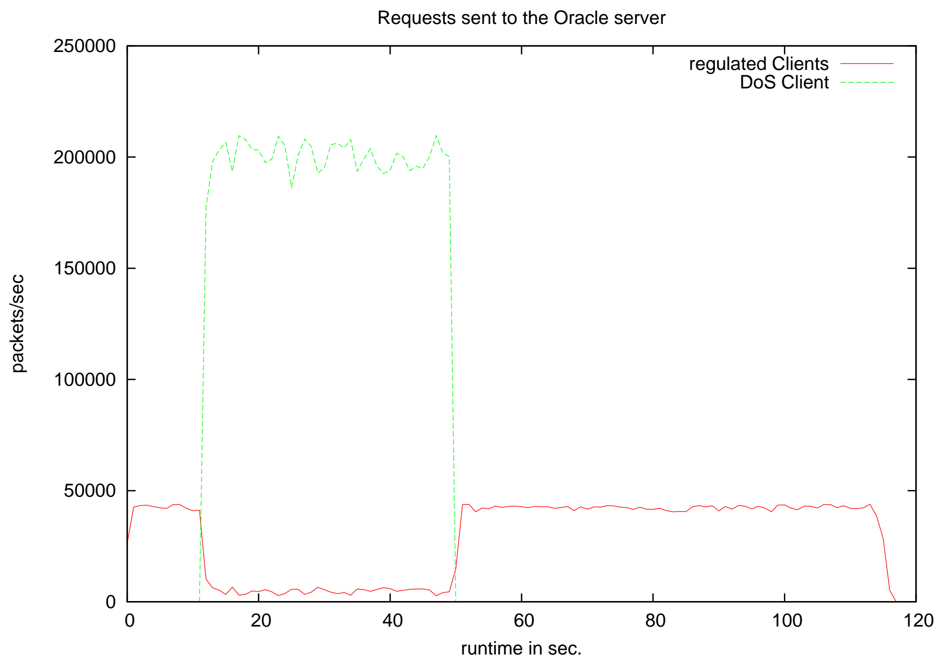

Figure 20: Queries sent to an Oracle server from regulated and flooding clients

Figure 21: replies from an Oracle server to regulated and flooding clients

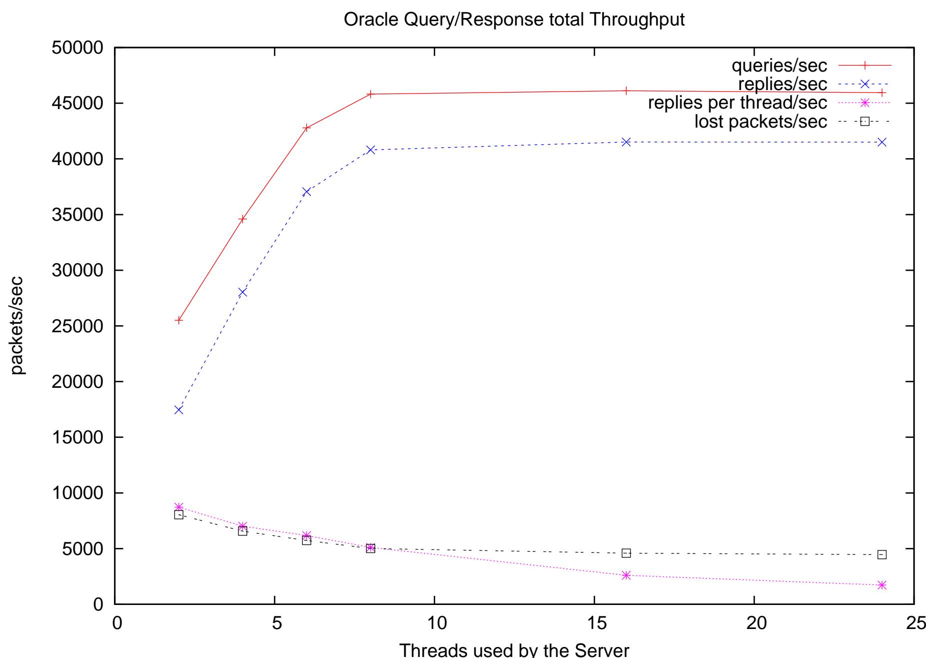

Figure 22: Query/Response behaviour with 100 IPs per request

hardware has a throughput of at least 80 megabytes per second as shown before, while this test never exceeded 35 megabytes/second.

Once the Oracle uses all CPUs the hardware offers, the reply performance no longer increases, but remains at the same level, no matter if a CPU is used by one, two or three threads. This indicated that it is the hardware resources which are missing rather than the threads influencing each other's performance.

One more thing that can be seen is that the amount of replies per CPU decreases, the more CPUs are being used by the server. While using two threads (and therefore two CPUs), each CPU manages to handle about 8,500 answers per second, this number decreases to only 7,000 per CPU when using four Threads and lowers even further to about 5,100 answers per CPU per second when using eight threads.

The first reason for the slowdown of the threads per CPU is that only one Thread can read a packet from the socket at a time, while others might have to wait a few CPU cycles until the socket is freed. Another reason why the performance is not increasing linearly is that the more processes are running under heavy load, the more software interrupts are used by the operating system. Thus it takes more time to handle the network cards and the socket the Oracle server uses.

A third reason for the slowdown is that the Threads are not exclusively assigned to one CPU and can roam freely between the cores and processors. As this means that the cache of the processors is put under more stress, it produces more cache misses and therefore increases the time it takes to read data from the RAM. Since the Oracle holds its entire database in the RAM, this can lead to a bottleneck, also slowing down the speed with which queries can be answered.

6.2.1 Packet loss

As can be seen in figure 22 there are more queries sent to the server than there are replies from the server. This indicates that some requests are lost on the way to or from the server. As the network is not a likely cause in this scenario, the packet loss must be taking place in the server itself. In this case, it is specifically introduced by the socket queue of the operating system the server is running on. If the queries

come in faster than the server can handle them, the queue will run full and overflow. Once the queue is full, newly arriving packets will be dropped by the kernel while the Oracle never became aware that they were ever there. This is not the behaviour of the Oracle but of the operating system it is running on.

As the clients are not synchronized with one another, there is on the one hand the possibility of packet bursts arriving at the Oracle, while on the other hand there is also a chance that there are no packets being received for several milliseconds. This results in fluctuation in the filling of the socket queue making it possible that even though the server is running over its capabilities, the queue is temporarily empty.

6.3 Round-Trip-Times

Besides the total amount of queries the server can handle per second, it is also important to know how long it takes to actually push out a response to a request. Even with 41000 replies per second, it does not say how long a client has to wait for the answer. This criterion is measured by the round-trip-time (or rtt). The rtt is the time from sending a request to the server to receiving a response.

When measuring round-trip-times, two different scenarios have to be evaluated. One is the minimal rtt a client can achieve. This indicates how long it takes the server to process a query plus the time the packet spends on the link. This type of round-trip-time will tell how the server can behave when the load is minimal to medium.

The second type of rtt is how fast the server can react when it is under heavy load. Here the minimal round-trip-time will not be achieved in most cases. Therefore it is more interesting to know what the average response time of the server is and how much it fluctuates. Thus, the second rtt is the average response time as well as the standard deviation from the average over several repetitions of the tests.

The link in the test scenario was not congested, introduced no packet loss and added an approximate 0.1 milliseconds to the round-trip-time. Figure 23 shows the rtt of the Server in respect to the number of used server threads. However, the minimal time is taken from the same test results, therefore being probably higher than it should be, as the server was constantly under heavy load.

Analyzing Figure 23 reveals that there is a minimum in the average rtt when each thread has one core/CPU to itself. This is due to the fact that there is no overhead for switching between processes that would slow down the calculation process, as well as no shared CPU for multiple threads. One more thing that can be observed is that the average round-trip-time does not increase dramatically when using multiple threads per CPU. This would suggest that once a thread has successfully received a query from the socket, it is barely interrupted by other threads. If this was the case, the rtt should increase by about two if each process only gets half the processing time. Likewise, the time should triple when a thread only gets a third of a CPU. However, this is not the case, as the CPU-time, after the tests were done running, was always evenly distributed among the threads.

One more observation is that more threads are decreasing the average response time drastically. When using two threads, the round-trip-time is at 7.55 milliseconds, while eight threads decrease the rtt to 2.6 milliseconds. The reason for the rtt decreasing with more threads available is that the queue of the network socket is more rapidly processed. If a request is queued in an almost full queue, it will take two threads a considerably longer amount of time to process all queries queued before it than it takes eight or sixteen threads. Decreasing the queue length will reduce the rtt for servers with only a few threads at their disposal, but will also increase the effects of packet loss as well as make the server drop more packets if bursts occur.

What can also be seen is that the minimal rtt the server has does not change as the number of threads changes. This indicates that if there is no load on the server it will always respond much faster than a server that is experiencing heavy load. It also shows that the threads are not cooperating to handle one request but run parallel, as cooperation between threads would decrease the minimal time for the rtt by the amount of threads that are being used.

The last thing that can be seen is the standard deviation of the average rtt. This measures how many milliseconds a single packet is off the average rtt in general. The higher the deviation is, the more fluctuation is within the results and the less reliable the server is to achieve the average response time.

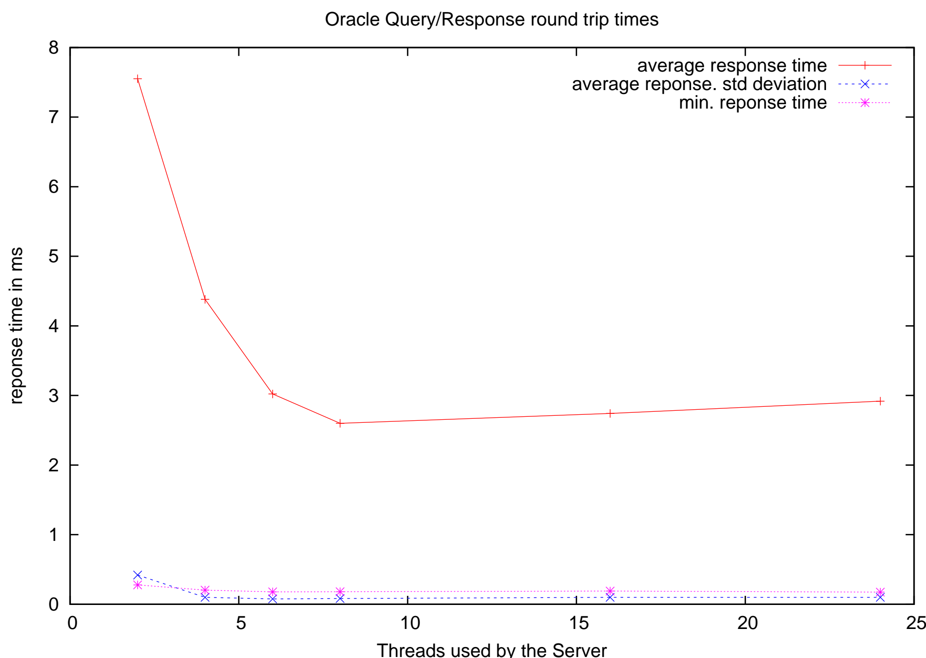

Figure 23: Response times in milliseconds of the Oracle Server during heavy load

As can be seen in figure 23 the standard deviation of the average response time is lower than the minimal response time. It is actually around 0.1 milliseconds, which is lower than the delay on the link. This indicates that if the server responds, the answer can be expected to be returned within 3 milliseconds when using more than four threads. Only the deviation for the Oracle using two threads is high, but seen in relation to the very high average response time, this is still a solid result.

In a real world scenario, the round-trip-time will be a lot higher depending on the latency the client has to the server. Therefore, the calculation time of about 2.5 ms the server needs to reorder the possible candidates in the query can be neglected when looking at a real world example.

6.4 Varying request sizes

The size of the request is an important matter when it comes to response times from the Oracle, as well as how many queries the server can handle. This section will focus on showing how the server behaves under heavy load. The server is always being queried over its maximum capacity.

6.4.1 Replies per second

Figure 24 shows how the Oracle server running with different threads behaves when the size of the requests changes. The range is from ten to 363 IPs per query. The reason for the lower limit, is that it is unlikely that clients send less than ten IPs to the Oracle as most overlay networks will have larger lists of potential candidates to be sorted. The upper limit is defined by the protocol overhead plus the payload size which should not exceed 1,500 bytes. Therefore 363 IPs is the defined maximum the server can handle in a single UDP packet.

One thing that can be seen in figure 24 is that the number of queries the server can answer is directly dependent on the size of the request. This is also supported by the fact that the sorting in the Oracle server is directly dependent on the size of the list containing the IPs.

Although the amount of responses the server can push out degrades as the packet size grows, it

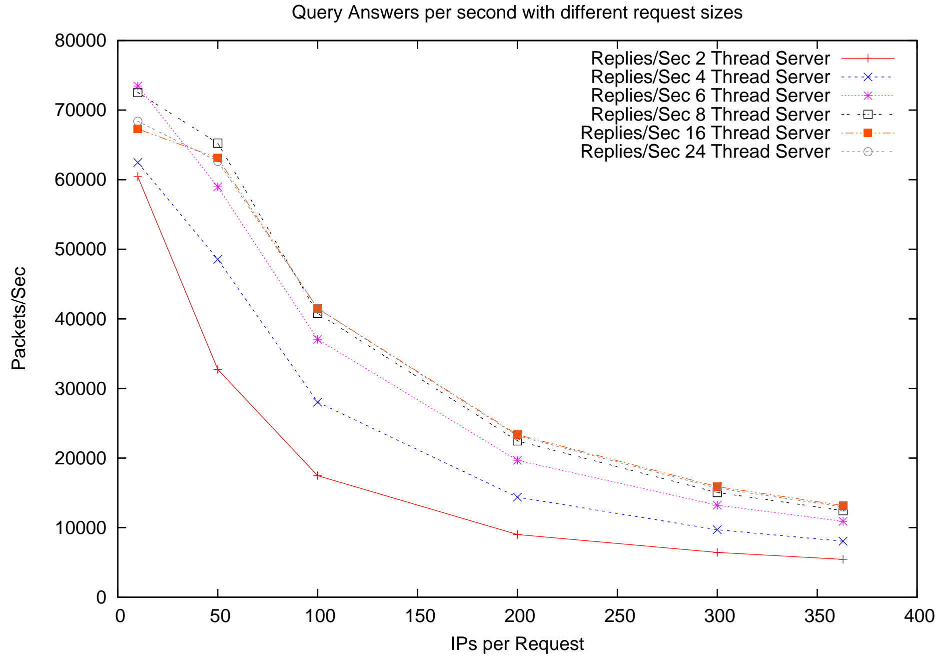

Figure 24: Replies per second of the Oracle server when changing the size if the requests

decreases linearly. Therefore, the Oracle is able to answer half the requests when the packet size doubles. This can be seen when observing packet sizes of 50 IPs or higher. This behaviour is due to the fact that most of the time for processing a request is spent on the content of the packet itself. This means that the information gathering as well as the sorting algorithm determines the behaviour and speed of the processing. Since information gathering in the Oracle is a $O(1)$ operation per IP in the request and the sorting has a complexity of $O(n * log(n))$ per IP list, a behaviour of an almost linear decline was to be expected.

However, a closer look at the behaviour of very small requests reveals the performance does not scale as well as it does with the other sizes. This is mainly caused by the overhead the protocol and the OS impose on the Oracle. While larger requests mainly spend time working on the content of the request, the small queries spend most of their time going from the socket to the Oracle and vice versa as well as being parsed and generated. These operations can be neglected with bigger packets, but make up an increasing amount of CPU time in the case of small packets.

It can also be seen that the servers running with fewer threads than available CPUs perform better in the case of small packet sizes. This is due to the high amount of packets streaming in, which takes processing power away from the Oracle, as the operating system needs this CPU time to handle the packets themselves. In the case of fewer but larger packets, the queuing in the socket takes over and the servers with more threads are able to perform better, as they are now able to use the full potential of all CPUs. This is due to the fact that the operating system itself needs less CPU time to handle the network cards as the packet size increases. Another reason is that when the packets increase in size, fewer queries are sent towards the server, since the clients reduce their sending speed due to a higher packet loss.

6.4.2 Round-Trip-Times

The simple throughput of the server is not enough to determine how good the implementation really is. Figure 25 shows the average round-trip-times a request takes to comes back to the client that sent it.

The main result that can be derived from figure 25 is that the average response time increases with the number of IPs that are sent in a request. As was argued before, this has to do with the fact that the Oracle has to perform lookups on every single IP address in the packet, thus increasing the time it takes for a query to be processed as it gets larger.

What can also be seen is that looking at very small requests and having most but not all processors occupied by the Oracle, this setup becomes more effective than giving the Oracle all the processors which are available. This is because of the CPU-time the operating system needs to handle the network cards. Therefore, the best rtt performance with small requests can be achieved, in this case, by using six threads.

As the number of threads increases, so does the rtt, as the operating system now has to share its resources with the Oracle, slowing down the handling of the queue of the network cards. However, these are averaged results. Looking at the standard deviation of these results in figure 26 it can be seen that those servers achieving the best average rtt with small requests are also the ones with the highest deviation in the round-trip-time. This lets the rtt fluctuate more, making it less likely to achieve the shown average rtt for a single request.

Looking at the results in figure 25 to see what server achieves the best performance when being queried by large requests, the lowest rtt is found when 16 threads are used. Marginally slower are the Oracle instances using eight and 24 threads. However, figure 26 shows that the standard deviation in the servers with the most threads is up to four times higher compared to the eight threaded Oracle. These round-trip-times, although better, fluctuate more and are therefore less reliably achieved. Taking the fluctuation into account, it is safe to say that the average round trip time is very close, if not the same in a real world application if the server is using all CPUs offered by the hardware.

Considering the round-trip-time in relation to the CPUs the server has available, it can be seen that the more threads the server has, the less time it needs to answer a query. This has nothing to do with the fact that the server is running slower or the threads work with each other, but is caused by the network socket queue. Since there are less threads available to empty the queue, requests that make it into it will always be added at the end. As it takes two threads considerably longer to process the same amount of queries and therefore to work through the queue of the socket, the time the packet spends waiting to be handled is increased. Therefore, with fewer threads, the round-trip-time increases because requests spend a longer time waiting to be processed.

One more thing that can be seen in figure 26 is that the deviation of the four threaded server at 200 IPs per request spikes unnaturally high. This can only be explained by the fact that something within the setup interfered with this test. However high this spike might be, the rest of the results fit so this was a one time anomaly.

6.5 Flooding behaviour

This test scenario was designed to examine the server's capabilities of handling maximum amounts of requests regardless of its capacity. As the client no longer wait for responses, but generate packets as fast as possible to push them towards the server, it is no longer feasible for the client to count reliably how many answers are returned. Also, in flooding mode, the client no longer spawns multiple instances of itself, but uses one thread to send out as many queries as possible.

In order to still be able to count the answers, tcpdump **??** was used to count the answers the server sent back. The dump of tcpdump was directly written to */dev/null* while only the pure statistics of how many packets were received and dropped was kept. The server was run with eight threads only for these tests and each test series was repeated 100 times.

Figure 27 shows the results of this test series. As the regulated clients do not send packets with only two IP addresses, there is no comparison possible on this smallest of all queries. What can be seen on the low side of packet size is that the regulated clients are receiving much fewer replies than their flooding counterparts. This is mainly due to the fact that the regulated clients have an upper limit to how fast they can send how much, while the flooding client does not care if the server is overloaded. Therefore, in the case of the flooding, the queue of the server is always full and it will perform at its maximum speed.

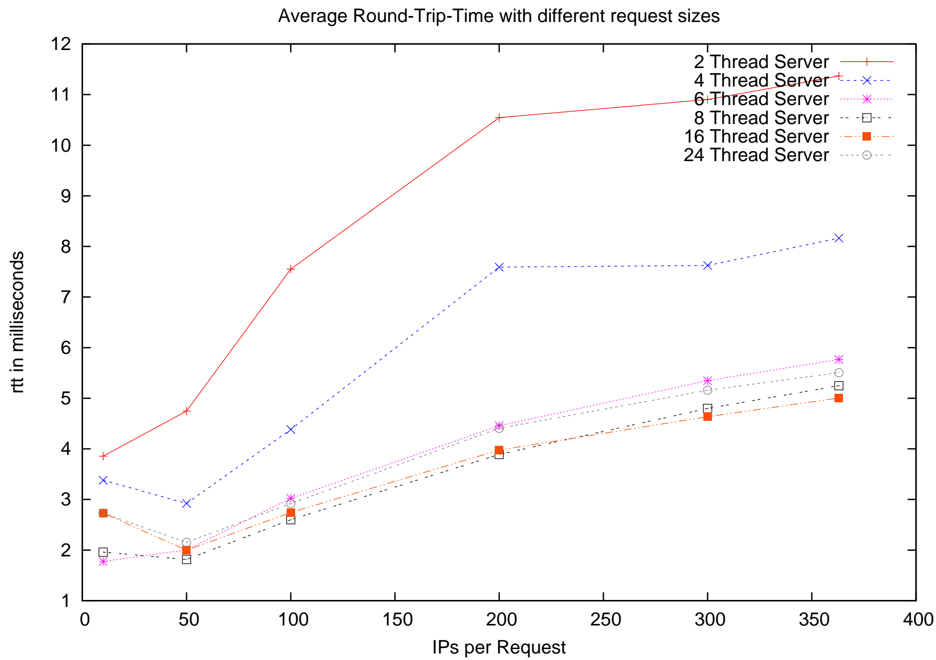

Figure 25: Average Round-Trip-Time of the Oracle server when changing the size of the requests

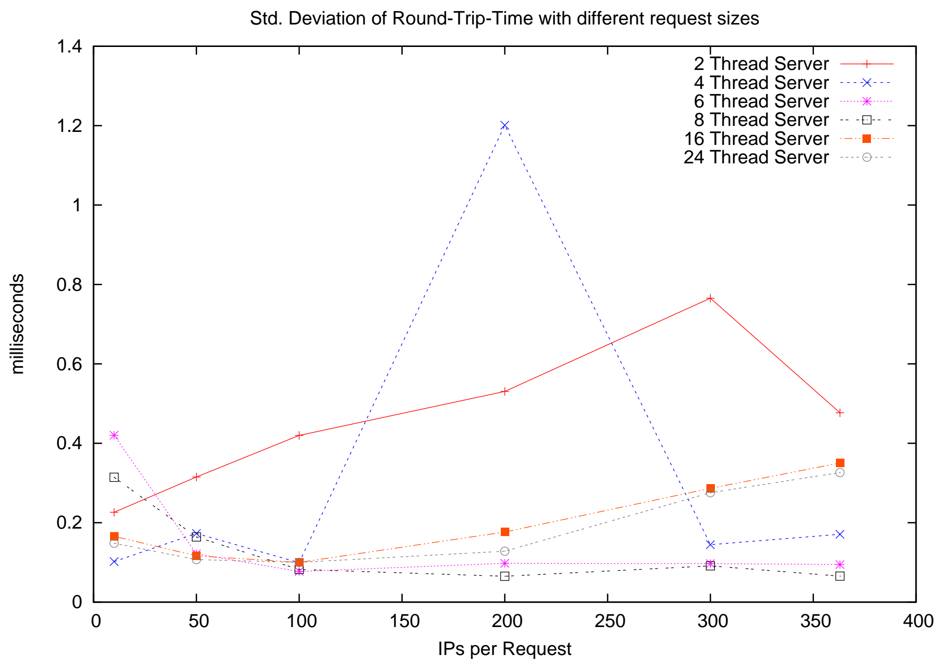

Figure 26: Standard deviation of Round-Trip-Times of the Oracle server when changing the size of the requests

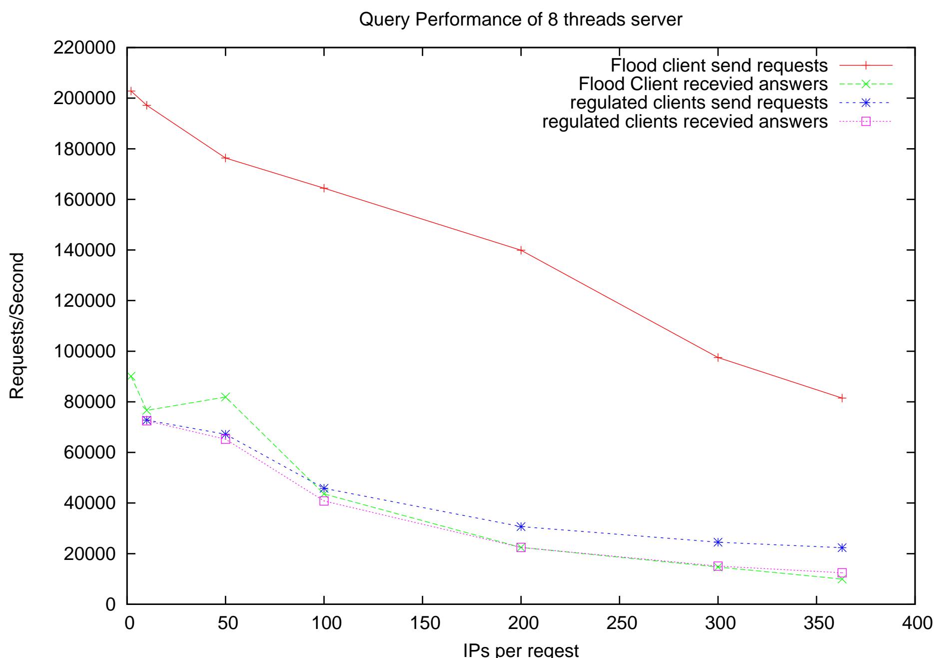

Figure 27: Query/Answer behaviour of an Oracle server using eight threads queried by regulated and unregulated clients

This also indicated that the regulated clients estimate the load the server can handle incorrectly at very low packet sizes, making the server look slower than it really is. What can also be seen at 10 IPs per packet is that the network cards are so busy handling packets, the performance of the server decreases due to it having to use a lot of resources to handle software interrupts. Here the server is mainly busy parsing and generating packets instead of performing the sorting it is required to do.

With increasing request sizes, the amount of queries per second is decreasing and the regulated clients are getting better at estimating the Oracle server's load capacity. At 100 IPs per query, the effect of temporary empty network socket queues can still be seen but is no longer significant.

When looking at large packets, the regulated clients are actually making the server perform better than the flooding client, as they are not overloading the network cards, taking away processing power from the Oracle and giving it to the handling of the software interrupts. However, even with approx. 80,000 requests a second, each yielding 1,468 Bytes, causing a network load of 30 megabytes/second filled with requests, the Oracle still manages to process 10,000 queries per second.
Therefore, it is safe to assume that even under very heavy load, the server is able to still answer queries and does not get overloaded with packet handling.

6.6 Small vs. big internal network

One of the requirements of the Oracle server from section 3.1 was that the size of the internal network should have no effect on the performance of the server. Figure 28 shows the query/reply results of the same test series run against an Oracle configured with the small network shown in figure 19 and then the same series re-run against the same Oracle with the larger networks shown in figure 18 loaded into it.

As can be seen, the results are mostly identical. This indicates that the network loaded into the Oracle does not affect the amount of queries the server can handle per second. Only when it comes to very small packet sizes, there is a difference between the smaller and the larger network. Figure 29 shows

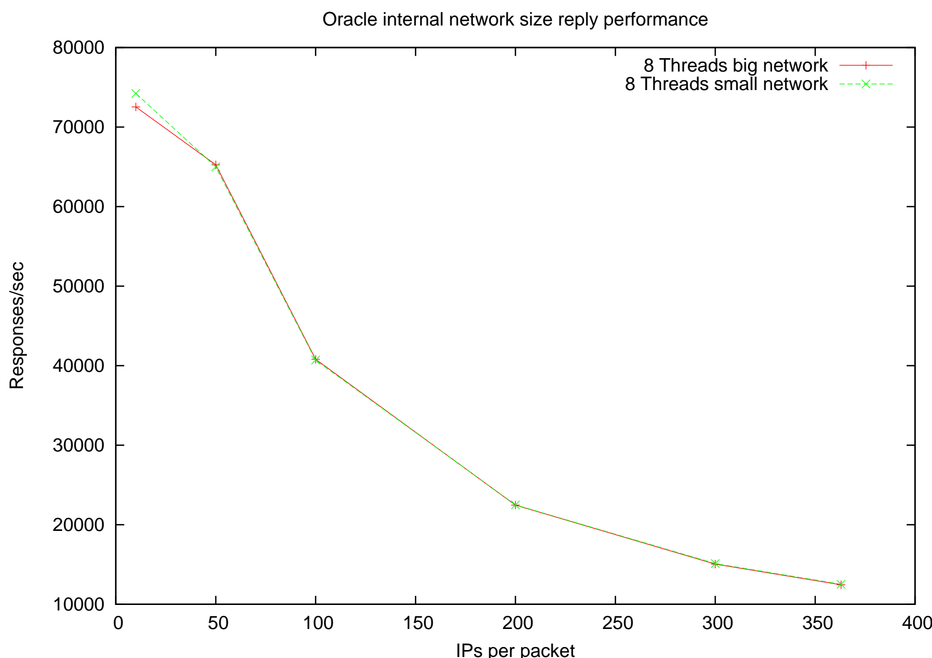

Figure 28: Query/Answer behaviour of an Oracle server using eight threads run with different internal network sizes

the results of the average round-trip-times of packets of the same test series. Again, the graphs are almost identical to each other except in the case of the very small packet sizes. The split that can be seen at the large packet sizes is a difference of about 0.03 milliseconds in the average. Since the deviation of the results is at 0.06 milliseconds. This spilt is considered to be just fluctuation within the tests and holds no indication that the Oracle with the smaller network is starting to perform better at larger packet sizes.

This leaves only the difference the results show at the small end of the scale to be discussed. Unfortunately there is no explanation for this result with the code of the software, as the rest of the results indicate the exact opposite of this one point. Also, all tests were run in batch mode straight after another, so this can also not be a change in the source code causing this difference between the two network sizes. The only explanation that can be offered for this anomaly is that the network the tests were run on was congested when the larger network was running, while the smaller experienced no congestion.

Despite this anomaly, it is safe to say that the size as well as the diameter of the internal network does not affect the performance of the Oracle server.

6.7 Stability

During the development of the Oracle server, it was continuously tested for memory leaks and possible invalid memory read/write accesses. While memory leaks can be found using Valgrind, it is harder to find invalid memory references in the software, as they lay dormant and do not cause any problems until they are read. Therefore it is never absolutely safe to assume that such a mistake is not lying dormant.

Table 6.7 shows the CPU-times spent on some of the test series that were run with the current version of the Oracle server. Unfortunately, some of the statistics are missing, so this is a lower estimate on the CPU time the server spent running without a crash.

What should also be noted is that the server cannot change its amount of threads while it is running. Therefore each displayed time needs to be interpreted as one run, as a change in the thread number also means restarting the Oracle server.

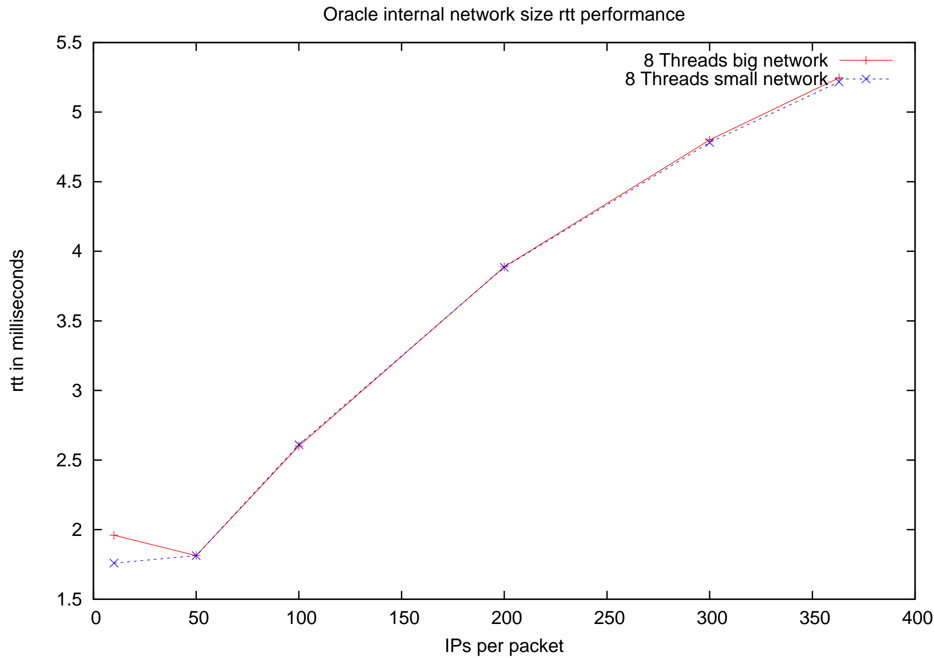

Figure 29: Round trip time for queries run with an eight thread server with different internal network sizes

However, when it comes to CPU-time spent by the server it can be said that the server, in its longest stretch, spent 138 hours without crashing or showing a degradation of its behaviour. This is no certainty that the server really is stable to the point that it can no longer crash, but it is a good indication that even under heavy load, the developed software within this thesis is stable in its current state.

6.8 Test result summary

The tests run against the server show that the server performs well in terms of round trip time as well as the amount of replies it can push out per second. With a peak of about 41,500 replies per second on queries that contain 100 IP addresses, it shows that the Oracle server can handle a fair number of clients. It was also shown that the multi-threaded architecture of the server considerably boosts the performance of the server when more CPUs are available. This is also valid when the number of threads exceeds the

Number of Threads	Total CPU Time	average CPU-times per thread
2	62h:39m	31h:19m
4	72h:34m	18h:08m
6	78h:47m	13h:07m
8	138h:28m	17h:18m
8s	95h:18m	11h:54m
16	82h:14m	05h:08m
Total	529h:51m	——

Table 5: CPU time spent during tests

number of available CPUs.

Furthermore, the results concerning the different network sizes are encouraging enough to claim that the network size and diameter of the internal network in the Oracle database do not affect the performance of the software.

7 Conclusion

The main focus of this thesis is on the implementation of the Oracle server. We envisioned that this service can be offered by an ISP to its customers to both improve end-user experience and enable effective traffic engineering by the ISP. The Oracle arranges IP addresses relatively, according to their network proxomity to each other.

The Oracle server implementation presented throughout this thesis is scalable, flexible and memory efficient. It is based on a multi-threaded achitecture that boosts its performance, and on an abstraction of the network topology to optimize memory consumption.

The chosen programming language for the Oracle server implementation is C++. The tools Valgrind and Callgrind were used extensively throughout the developmental process to ensure the highest quality of the service with regards to speed of handling queries and most effective utilization of hardware resources. Due to the modularity introduced to the software, the Oracle server can be easily modified to adapt its ranking function to all requirements, regardless of the clients served and the underlying network topology. The communication between client and server is handled by one of the first versions of the flexible Proxidor protocol.

To evaluate the performance of the Oracle server, a test client was incorporated into it. In its simplest form, the client replicates the behaviour of a ping. The advanced functions are capable of spawning numerous processes to simulate multiple clients in regards to simultaneous querying or performing flood attacks on the server. These features enable a wide range of test scenarios to be performed on the Oracle server.

The experimental part of the thesis focuses on three issues. First, the maximum throughput of the software is tested in respect to the size of the queries as well as the number of threads used by the Oracle server. Second, the round-trip-time of requests is evaluated using the same parameters as previously stated. Third, flooding attacks are run against the Oracle server to assess its stress resistance and recovery time.

The results reveal an increasing throughput of replies in relation to the threads utilized by the Oracle server. The answered requests per second employing 16 threads were observed to exceed 41,500. While being under full stress, the Oracle server achieves an average round-trip-time of 2.6 milliseconds per reply. Evaluating the behaviour of the Oracle server during DoS attacks shows a decreased response rate. As the attack ceases, the Oracle server recovers within one second to its full capabilities.

Supported by the presented results, we conclude that the developed Oracle server will assist ISPs to better engineer their traffic. With its DNS-based structure, fast internal database and highly flexible protocol, the Oracle server is easy to integrate and deploy.

7.1 Future Work

The concept of the Oracle server developed in this thesis provides a multitude of possibilities for further development. Not least of which is the evolution of the software into a fully featured inter-AS Oracle server. Having multiple ISPs working unanimously will further maximize the effect the server has on traffic engineering and improving user experience. The Oracle servers must then exchange aggregated subnet information via the Proxidor protocol while staying secure, fast and keeping protocol overhead to a minimum. Therefore development has to be concentrated on improving the Oracle database, the protocol handling within the server and aggregating information most effectively for other cooperating Oracle servers.

A more theoretical approach is looking at different ranking functions used by the Oracle. This field aims at the optimization of the ranking function, taking different overlay network structure into account. The aim is to analyze popular overlay networks and incorporate different classification techniques to continually improve the spectrum of services provided by the Oracle server.

References

[1] Vinay Aggarwal, Obi Akonjang, and Anja Feldmann. Improving User and ISP Experience through ISP-aided P2P Locality. In *Proceedings of 11th IEEE Global Internet Symposium 2008 (GI '08)*, Washington, DC, USA, April 2008. IEEE Computer Society.

[2] Obi Akonjang, Vinay Aggarwal, Anja Feldmann, and Georgios Smaragdakis. The Oracle Protocol Draft v0.2. Proxidor Protocol draft Version 0.2, Jan 2009.

[3] Ruchir Bindal, Pei Cao, William Chan, Jan Medved, George Suwala, Tony Bates, and Amy Zhang. Improving Traffic Locality in BitTorrent via Biased Neighbor Selection. *Distributed Computing Systems, International Conference on*, page 66, 2006.

[4] David R. Choffnes and Fabián E. Bustamante. Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 363–374, New York, NY, USA, 2008. ACM.

[5] KCachegrind. `http://kcachegrind.sourceforge.net/html/Home.html`.

[6] KDevelop. `http://www.kdevelop.org`.

[7] Nikolaos Laoutaris, Georgios Smaragdakis, Azer Bestavros, and John W. Byers. Implications of Selfish Neighbor Selection in Overlay Networks. In *Proceedings of IEEE INFOCOM 2007*, Anchorage, AK, May 2007.

[8] The Linux man-pages project. `http://www.kernel.org/doc/man-pages`.

[9] socket - Linux socket interface. `http://www.kernel.org/doc/man-pages/online/pages/man7/socket.7.html`.

[10] University of Oregon Route Views Project, Jan 2009. `http://www.routeviews.org`.

[11] Osama Saleh and Mohamed Hefeeda. Modeling and Caching of Peer-to-Peer Traffic. *Network Protocols, IEEE International Conference on*, pages 249–258, 2006.

[12] Skype. `http://skype.com`.

[13] Y. Takefuji. Can We Survive an Internet Blackout? [Letter]. *Technology and Society Magazine, IEEE*, 26(3):8–8, Fall 2007.

[14] tcpdump. `http://ee.lbl.gov`.

[15] Valgrind. `http://valgrind.org`.

[16] Haiyong Xie, Yang Richard Yang, Arvind Krishnamurthy, Yanbin Liu, and Avi Silberschatz. P4P: Provider Portal for Applications. In *Proceedings of ACM SIGCOMM*, [Seattle, WA], August 2008.