Erlydns

A DNS server written in Erlang

Max Amanshauser
(max at lambdalifting dot org)

December 1, 2008

Abstract

Erlydns is a new DNS server written for a university project at TU Wien. It's purpose is to explore Erlang, an interpreted, functional, parallel-computing language, as a tool to write secure and fast services.

1 Motivation

Traditionally DNS servers and services in general are written in native-code compiled languages without runtime system or automatic memory management, such as C and C++. This can easily lead to certain security vulnerabilities due to the high attention to detail required. The underlying assumption was that modern interpreted languages are fast enough to write traditional network services in and that the advantages far outweigh any performance hit, like it happened in the area of dynamic web sites. A further goal was to spare the user the burden of learning zone files or similarly cryptic data structures as far as possible. Last but not least the author wanted to deepen his understanding of DNS, one of the most important and popular services on the internet.

2 Erlang

The strengths typically associated with native-code compiled languages are high performance and platform-independence. They do however have many problems solved by more modern languages, such as cumbersome manual memory management, low expressibility leading to low productivity, buffer overflows and similar security vulnerabilities caused by manual memory management. On the plus side are high performance and portability, both of which we will try to achieve using Erlang.

Erlang is a general-purpose concurrent programming language and runtime system. The sequential subset of Erlang is a functional language, with strict evaluation, single assignment, and dynamic typing. For concurrency it follows the Actor model. It was designed by Ericsson to support distributed, fault-tolerant, soft-real-time, non-stop applications. The Erlang distribution comes with an extensive library, which is the only external code used by Erlydns, containing useful tools like Mnesia, a distributed database.

Erlang's focus on concurrency encourages parallel programming, which is very easy in comparison to imperative languages, due to its message-passing, share-nothing style. The run-time system is sophisticated and features both user- and kernel-level threads to maximise use of modern multi-core processors.

Erlang nodes running on different machines can communicate in the same way as nodes on the same machine, hence computer boundaries become invisible.

3 Erlydns

3.1 Features

Erlydns is an authoritative DNS server, which means it does not answer arbitrary queries by recursing through the DNS system, but only those it has local data for. Such a server is for example used by hosting providers for their customers' domains. Erlydns tries to be a simple and easy to use server for individuals that just have a few domains, as well as scaling well to thousands of domains. Experience shows that users serving a large number of domains need to integrate their DNS server with the rest of their infrastructure, e.g. via a web interface for customers or an automatic mail gateway. The database modules written for the Erlydns software provide a convenient way to write such interfaces.

Outstanding features of Erlydns are:

- Database Distribution: Erlydns uses the built-in distributed Mnesia database to store domains and their associated Resource Records (RRs). This way only changes are, optionally via SSL, transferred between servers, compared to the less efficient AXFR. Adding a new server is done just by running a few lines of code.
- Erlydnsadmin utility to add new domains and records. The admin tool tries to perform sanity checks on user input and only accept valid data.
- Erlydns does away with the notion of master and slave servers. In traditional DNS servers one server per domain is designated to hold the current data (master), all other servers regularly poll (slaves). When resolving, however, all of the servers can be queried and are therefore at risk of attack, so with Erlydns every server is master/slave at once, allowing new data to be entered on any server which is afterwards mirrored to all the others.

3.2 Security

Erlydns was written with security in mind. Examples of deliberate steps taken to increase security are:

- No dependence on third-party libraries
- No root privileges required at any time. Experience shows that even when privileges are dropped at the earliest possible moment, services can still be exploited before that point. If the user wants to run the server on a privileged port on a system which does not allow this for ordinary users, it is recommended to use the hosts' OS to redirect the port.
- To insure against bugs in Erlydns that could allow a straightforward DoS attack a new process is created for every incoming request. The code before this "fork" is merely about 15 lines.

Due to its dependence on Erlang, Erlydns shares its primary weakness:

- The highly distributed nature of Erlang enables remote code execution on any Erlang node. The only security measure is a pre-shared key. Therefore it is imperative that every Erlydns installation additionally restricts access to the Erlang ports to trusted hosts.

4 Design

To ensure consistent nomenclature we call a bit-level representation of DNS traffic a *packet*, when the DNS data resides in Erlang data structures we call it a *message*.

The top module is **server**, which opens the port and spawns new processes with the function **queryhandler:loop/0**. **queryhandler** is the logical top module for one individual request. It analyses the packet, stores its contents in a message, performs the appropriate action, creates a response packet and sends it.

The individual modules involved in this process are:

- **lib_mnesia**: This library module contains all necessary functions to read and write from/to the database, including several complex queries like “What RRs exist for a given domain?”, or “Find all domains ending with a given suffix.”.

- **lib_packet**: This library module is responsible for all actions regarding conversion between messages and packets.

- **lib_string**: This library module contains functions for converting between Erlang strings and DNS domain names.

- **suffix_server**: DNS has a primitive compression algorithm. If the length octet of a DNS label (a domain name consists of several labels) starts with two consecutive 1s then the next 14 bits are not part of any character but are an offset into the DNS packet. The RFC then requires reading from the offset up to the end of the domain name. **suffix_server** is spawned by **queryhandler**. All labels read are then stored with the server together with their offset. If an offset is detected during initial parsing, the server is queried for the corresponding label.

The remaining relevant files of the Erlydns distribution are:

- **erlydnsadmin**: This module contains the admin tool.

- **inet_dns.hrl**: This file contains very simple macros in order to being able to use symbolic names in the code instead of bulky DNS constants.

- **records.hrl**: Contains all custom record definitions used in Erlydns, including those used in the database as data types.

5 Performance

All benchmarks have been performed with a parallel query system, sending 20000 consecutive queries on a AMD Athlon^{TM} 64 X2 Processor 3800+.

The results given in Table 1 (all values: queries/sec) were obtained by asking for a domain that the server is not authoritative for. Quite astonishing was the fact that erlydns actually performed **worse** in absolute terms when given more processors. The most likely reason is Mnesia’s locking mechanism as it acquires a global table lock on every query making concurrency all but impossible. The overhead for managing these locks explains the moderate decrease in performance. Also of note is the tremendous performance increase seen in later runs of powerdns, which is indicative of caching.

Table 2 reinforces the belief that Mnesia is at fault for poor performance, since erlydns only performs one Mnesia query searching for an A record, but several more when delegation with glue occurs. Powerdns show consistent results regardless of the nature of the query.

On the upside erlydns in its current, entirely unoptimized state performed sufficiently well when answering records directly, at about 30%

of powerdns' speed in our benchmarks. The vast majority of queries in practice will be of this kind, so practicality of Erlang network services is certainly achievable.

Topmost priority must be to remove the bottleneck in the database and allow concurrent queries. Several ideas exist to do this in a future version. Secondly the search in the database needs to be optimized such that delegation does not cause a performance hit.

6 Challenges

6.1 RFC issues

The DNS is over 20 years old and has since then often be extended by releasing new RFCs. Today there are 114 DNS-related RFCs, plus many obsolete ones. The main RFCs, RFC1034 and RFC1035, are partly obsolete and incomplete. An example for obsolescence is the support of networks besides the Internet, which do not exist anymore today. An example for incompleteness is the unspecified behaviour when a server is queried for a domain it is not authoritative for. *BIND* responds with a list of root servers, *Tinydns* does not answer at all, *PowerDNS* responds with SERVFAIL (Erlydns mimics *PowerDNS* here). Partially the RFC is intentionally ignored today, because adhering to it would have unacceptable results. As an example the resolve algorithm proposed in RFC1034 is prone to DNS cache poisoning.

Obviously a single coherent specification, to which any DNS product can be compared to, would be beneficial.

6.2 Statelessness

Since Erlang is a functional language some intrinsically stateful parts, in particular the admin tool, were relatively cumbersome to write compared to an imperative language like Python.

7 Outlook

Converting the entire software package to the OTP framework, which provides a standard way to perform process monitoring, hot-code-replacement, etc. would be beneficial in the long run. The user interface could still be improved, ideally almost no knowledge about DNS internals should be required. One obvious extension is recursion. The recent poisoning attacks on many popular DNS servers have spiked interest in DNSSEC, which could also be added to Erlydns.

Table 1: Five runs for 20000 queries

erlydns	erlydns -smp	erlydns -hipe	erlydns -hipe -smp	powerdns
2463.66	1702.42	2545.18	1486.88	5493.00
2498.75	1637.06	2508.47	1675.18	6195.79
2513.19	1631.06	2510.36	1541.66	6975.93
2575.99	1706.19	2545.82	1537.99	7757.95
2475.25	1604.75	2298.59	1581.03	9411.76

Table 2: Benchmarks for several different query kinds

	erlydns	erlydns -smp	powerdns
not authoritative	2182.93	2133.10	14084.51
A record	3900.92	3869.22	13642.56
NXDOMAIN	1189.70	1177.16	14306.15
redirected w/ glue	825.59	833.02	13879.25