Deploying (and Developing) Free Software for Network Administration

Alexios Zavras
zvr@aueb.gr

Network Operations Center
Athens University of Economics and Business
Patission 76, GR-104 34 Athens, Greece

1. Introduction

The Network Operation Center of the Athens University of Economics and Business (AUEB) is responsible for the system and network management of the main servers and the moderately-sized but rather complex network of the University. This paper presents some general thoughts on the state of network administration tools, based on our experience. It also describes two systems that we developed in our facility, in order to help us manage two specific areas of our responsibility.

2. Network Management Tools

The environment for system and network administration today is composed by heterogeneous computing systems supporting multi-vendor applications upon a variety of underlying transmission facilities and switching systems. Nowadays, network administration problems and tasks often exceed, in resources needed to resolve them, the problems that occur in single systems. Therefore, there is need for tools that facilitate the solutions of such problems.

In order to manage such a complex environment, one needs tools that are simple yet powerful, user-friendly yet easily customizable, and as complete as possible yet easily interconnecting with others. Although the above specifications might look like a quest for something unobtainable, a combination of currently readily available tools and development of custom software based on freely available libraries can extend a long way towards this goal.

2.1 Commercial and freely available

An easy, yet important, way to categorize the available tools is according to their mode of distribution. The major differentiation occurs between commercial offerings and freely available tools, although the specific licensing terms in both cases can be greatly confusing.

As a general rule, commercially available tools tend to be large systems that handle many faces of the difficult tasks a network administrator should perform. The freely available packages, on the other hand, usually have a limited scope and they focus in a very small number of corresponding tasks.

Due to the immense complexity of the problem, it is easily understandable why a system oriented towards the near totality of problems would be large. Additionally, since its development requires considerable resources, the commercialization of the software could be thought of as the natural way to recover the costs and make a profit.

In the case of freely available tools, there seems to be a disproportionally small amount of network management tools. Traditionally, freely available tools were geared more towards programming, with a small minority of products developed for system administration. Network administration tools were inexistant for a long time, even after the rapid growth of network connected systems and the explosion of the web. The first significant number of such tools first appeared in the area of security checking, usually being extensions of tools running on a single system.

2.2 Applications and libraries

Another way of differentiating the available software products is by their design goals. Broadly speaking, network management tools can be either applications that perform one or more functions, or libraries that can serve as an ifrastructure for the development of applications.

A clear example is the `mtr` application, which is an enhanced version of the standard traceroute utility. On the other hand, `Tnm` which is part of the Scotty distribution, is a library that allows programs written in Tcl to access network management information sources using a variety of protocols like SNMP, ICMP, and DNS.

Of course, there are also some applications where this differentiation is not that clearly defined. As an example, `mrtg` is a complete application, but depending on the configuration, which is extremely flexible, can be made to perform very different tasks.

2.3 A third alternative

One of the inherent drawbacks in the field of network management tools is the enormous diversity of functional network environments. Without exaggeration, the specific components of each and every one of them, concerning systems and services offered, are different. While such differences might not be important for some system administration tasks, they are crucial for correctly designing, developing and deploying network administration tools. The consequence is that the tools that might be appropriate for one environment might be less than useful in another.

Faced with these findings, one should consider carefully the alternatives. On one side there are massive commercial offerings that require a lot of resources. On the other side, there are freely available tools which might not be suitable for the specific environment.

Finally, not to be lightly discarded, there is a third alternative, namely of producing custom tools based on already available libraries. In our experience, for a specific set of problems, this alternative worked out for the best.

The next two sections will describe two of our custom-made tools that were built to address specific needs.

In our environment, a large part of software development is performed in Tcl, so it was natural to build network tools based on Tcl and Tnm. Although Tcl and Tnm are freely available, they were by no means the only such software packages that were used for the development of the tools. In their current form, the tools make heavy use of a variety of libraries for infrastructure, like database management, or for convenience, like HTML generating code.

Of course, we should also not disregard the daily use of freely available operating systems, such as Linux, and software development tools, such as the FSF tools like `make`.

3. The Cordial System

The first example of a custom-made system that was developed for helping in the tasks of system and network administration that will be presented is the Cordial system for managing dial-up access.

The principal components of the system include a server for communicating with the access server, a database for storing user profile information, a database for storing connection logging information, and a reporting module based on web technology.

For all components, the existence of various freely available implementation proved crucial in the development of the integrated system.

3.1 Functional requirements

The purpose of creating the Cordial system was to have a robust, yet simple system for allowing all the members of our academic community (faculty, students and administrative staff).

One of the characteristics of our environment, which proved to be an advantage in the final implementation of the system, was the relatively small size of the user community. Therefore, a number of design decisions were made that would be not practical if the system were to be used, for example, in the operations of a nation-wide ISP.

Another requirement was that the system be very flexible in accommodating different categories of users. For example, members of the faculty should be able to have a different set of privileges than the one available for undergraduate students. After the initial design phase, it was decided that the only way to achieve this functionality was to allow for the inclusion and interpretation of almost arbitrary rule-based policies.

3.2 General overview

The block diagram of the different components of Cordial system is presented in Figure 1.

The incoming calls come through a number of ISDN Primary Rate Interfaces (PRIs) to an access server, which in our case is a Cisco 5300. The use of PRI lines allows for the accommodation of both traditional modems over analog lines and digital ISDN connections.

As soon as the modem connection is made, the authentication process asks for a username and a password. These are checked by comparing them to a central database, and if they are valid, the user can use the service.

After the user having authenticated himself, a separate process determines whether the authorization to connect should be granted. While the first step of authentication results in uniquely identifying the user, this second step takes into account information about the current state of the system as well as historical information of the user s past connections and reaches a decision.

Once the user is authorized to connect, all the relevant data are recorded for accounting purposes. This accounting log, which is also updated when the user disconnects, not only includes the mere fact of the user connection, but keeps

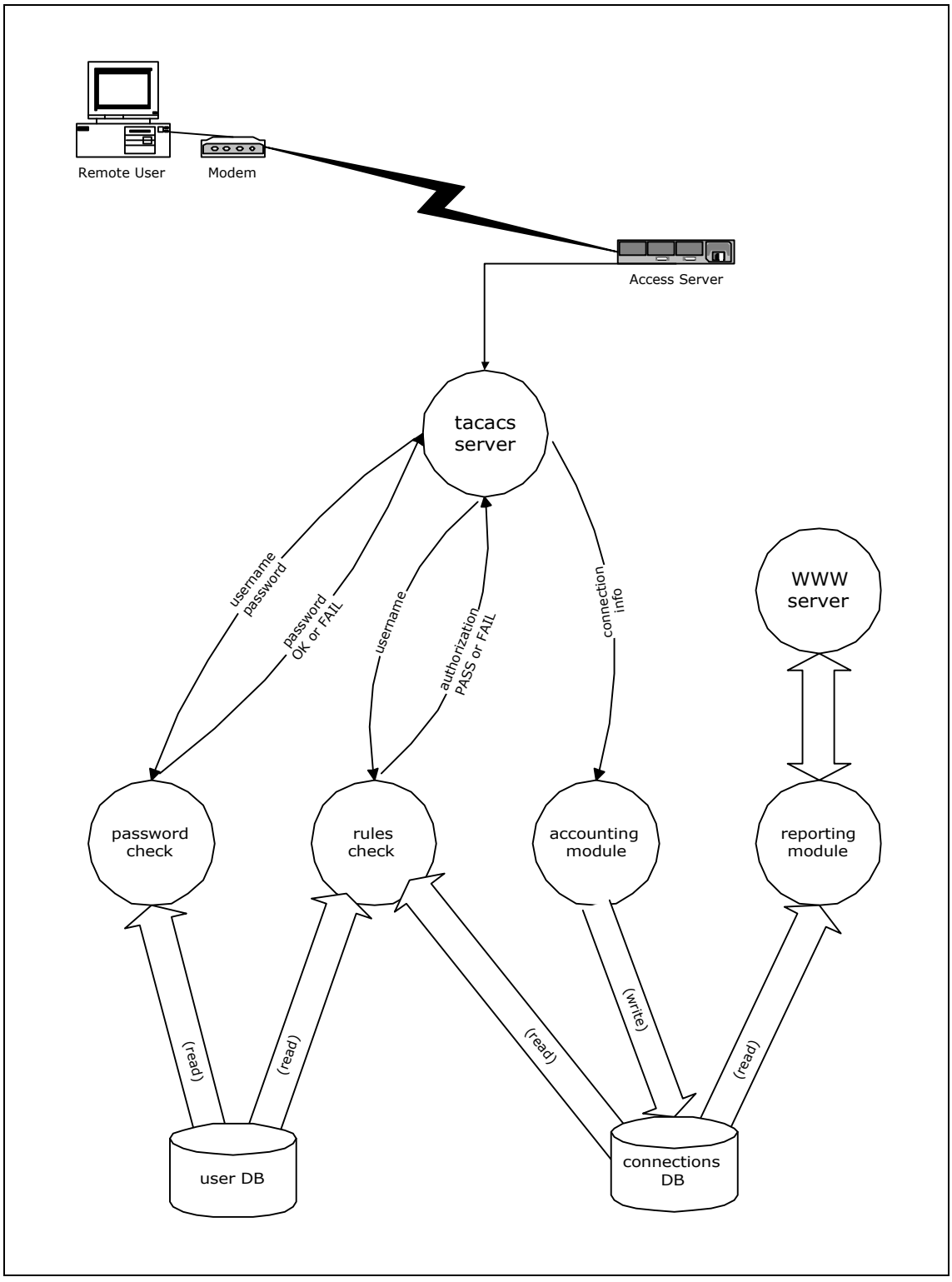

Figure 1: Cordial block diagram

also record of the amount the connection lasted, as well as the number of bytes and packets that were transferred from and to the user.

For the above triple functions of authentication, authorization, and accounting (AAA), the access server needs to communicate with different processes via a well-defined protocol. The two most widely used families of such protocols are TACACS and RADIUS. While the RADIUS solution is the official standard one, and the only one that is guaranteed to work on cross-platform environments, our setup could easily use the TACACS+ protocol, which was developed by Cisco. The source code for a TACACS+ server is freely available on the web, and its simplicity was a major advantage. In retrospect, this simplicity also proved limiting in some aspects where major changes were considered. However, the code is easily understood, and minor changes and enhancements can be easily added.

3.3 Authentication

The authentication module is responsible for checking that the caller is a legitimate user. The system had to be used by all the members of the academic community, even though a central repository of user information did not exist. The closest substitute that could be put to use were a number of Unix servers, that acted as mail servers and therefore had account information for large groups of users. For example, one such host handled all mail for undergraduate students. Needless to say, these servers were in different network, and, more importantly, different administrative domains.

Therefore it was decided that the account information should be obtained from these servers. Since all that was needed were usernames and passwords, the information on the Unix hosts were stored in the files `/etc/passwd` and `/etc/shadow`. Due to requirements for user categorization, the information on the groups each user belonged, were also needed at a later stage. This was handled in a similar manner, based on the `/etc/group` files on the Unix hosts.

A first prototype had the authentication process query the remote Unix host for verification of the password that the user had entered. This required a simple client-server protocol through a secure communication channel. However, due to the instability of some of the hosts and network connections, it was decided that the account information would be duplicated on the authenticating host, thus avoiding all network traffic for this process.

The duplication of the information is performed by a periodic transfer of the relevant files from the remote Unix hosts to the authenticating host. The information is then entered into a local database system.

Our institution is, slowly but steadily, moving towards deployment of an LDAP infrastructure, where all information concerning users will be readily available by means of a standard interface. The Cordial system has already all the hooks to query an LDAP server and obtain the user information that way.

3.4 Authorization

Once the user has been authenticated by means of a supplied password, the process of authorization is initiated. Since all that is permitted to the users is to initiate PPP connections, this step might at first be thought of as unnecessary. However, it is at this point that different policies about the use of the system can be enforced.

The authorization subsystem gets the authenticated username and consults a series of policies that apply to the specific user. Since most policies are group based, the first task of the subsystem is to associate the user with one or more groups. It then processes the rules which define the appropriate policies, and results in the correct outcome.

Since it was felt that the set of enforceable policies could not be accurately predefined, it was decided that the policies should be written as scripts in the scripting language Tcl, and allow their interpretation during run-time. The first step was to carefully define a small set of library functions that allow the policy scripts to gather information concerning the user and the environment. Based on this set, the policy scripts can be arbitrarily complex, although in practice they rarely exceed twenty lines of code.

As an example, a fictional policy for a faculty group can be seen in Figure 2. This policy disregards the username during the process of deciding whether the request should be authorized, and only uses it in order to calculate the amount of time the user has already used the system. This calculation is performed by a couple of library functions and is based on historical accounting data that were kept during the earlier uses of the system by the same user. The presented fictional policy imposes limits of one hour for daily use and of fifteen hours for monthly use.

```
# fictional rule for faculty group:
# allow logins, max 1 hour per day, 15 hours per month
proc faculty {username} {
    set already [day_usage $username]
    if {$already > [expr 1*60*60]} {
        puts "101 Over daily limit"
        exit
    }
    set already [month_usage $username]
    if {$already > [expr 15*60*60]} {
        puts "102 Over monthly limit"
        exit
    }
    puts "0"
}
```

Figure 2: Example policy

It should be noted that in real life, the policies are a little more complicated than the one presented here, since they also take into account other elements, such as the current time of day and day of week. They are also capable of returning more information that just a single numerical code with a text explanation, like the amount of time the user is permitted to use in this session. These parameters are passed back to and used by the access server which performs the actual PPP communication with the dial-up user. It is thus possible for the policies to specify information that can actually impact the capabilities of the user during the connection. A prime example of such measures is the invocations of access lists, which, for example, can prohibit the use of network application such as ICQ to the undergraduate students.

3.5 Accounting

The third and final part of the AAA subsystem is the one that performs all the accounting functions. This subsystem accepts the data from the access server and updates a database. The database structure is just a simple table where rows are constantly being added, without any accumulation taking place.

The accounting information for each connection, as mentioned before, includes information on the user, the time interval the connection was active, communication parameters such as the caller phone number and the IP address that was assigned, as well as traffic measurements in both packets and bytes in both directions. By keeping all the data, one has the ability not only to produce extremely detailed reports, but also to implement authorization policies based on these measurements.

3.6 Reports

The Cordial system can generate a number of different reports, according to the some user-specified criteria. The reporting subsystem, in order to be generally usable, was developed as an integration of a web-based front-end and a back end that was capable of querying the connections database. Almost all of the resulting reports are produced by aggregating the information that is kept for each user connection.

Besides the reports that describe the use of the system per user or in total, another series of reports is produced in answer to specific questions. An example of such a report is the list of users that were assigned a specific IP address during a time period, which can be helpful when investigating incidents of abuse if network services.

3.7 User database

The Cordial system uses mainly two databases for its operation. The first is storing all the information about the users, while the second one is storing all the information concerning the actual connections, i.e., the use of the system.

The user database contains the list of the valid users, their passwords, and the list of groups in which they belong. The data is stored indexed on usernames, since all accesses are being done with this key. The main use of the database occurs during the authentication process, where, given a username, the system, retrieves the associated password. A secondary use of the database is for reporting facilities, where, given the username, pieces of information such as the real name and phone numbers are retrieved.

As mentioned before, the user database is currently being populated by processing Unix account information files. It is worth noting that, for the size of data in our case, timing benchmarks showed that instead of comparing data and updating the database with only the modifications, it was much faster to completely overwrite the database with the new information.

In the future, with the deployment of the LDAP infrastructure, this database will no longer have a reason to exist on a local level, except as a caching mechanism for performance reasons.

3.8 Connections database

The connection database stores records for each connection, and is indexed by a unique identifying number for each connection.

The initial design called for a modification of the TACACS server to append the data to the database. However, since the server already prints this information to a log file, it was easier to merely process this accounting file and put the relevant information in the database by means of a separate process.

There are two types of queries where this database is regularly used. The first is, given a username, it should produce all the connections made by this user since a specified timestamp. This is used to calculate the accumulated data per user, which are in turn used in the different authorization policies. The second query reports the active connection for a specified timestamp and a client IP address, and is mainly used, as mentioned before, for tracking purposes in situations where abuse of network services was detected.

Apart from these simple queries, the database is also used to generate accumulated data for the production of statistical reports depicting the total use of the system.

3.9 Notes on database systems

Since there are many freely available database management systems, this section provides a little insight into a number of factors that were deemed important into choosing one of them.

The initial implementation of the database subsystem was based on SQL databases, using, in fact, in the early stages `mSQL` and later `MySQL` database systems. However, at a certain point, it was perceived that the functional requirements were simply calling for a database system which could store and get pieces of information, and no use was being made of the various structured query features of the SQL databases.

Therefore, a design decision was made and the above mentioned database systems were abandoned in the favor of the simpler but equal powerful Berkeley DB system. This database system provides a generic structure containing a key and data, which of course can accommodate whatever needs to be stored.

Besides the aforementioned connections database, which stores all the accounting data, a number of query databases are also used. These correspond roughly to indexes that are created for different columns of the same table in SQL database systems. Since the set of usually asked questions on the data is predefined, the definition of the appropriate fields to serve as keys in different databases was straightforward.

A significant advantage would come from using a database that could handle temporal data in an efficient way, since all entries in the connections database are flagged by a timestamp and correspond to a time interval where the connection was active. All timestamps are stored as the usual Unix `time` format, i.e., seconds since the beginning of 1970, and time intervals are specified by a pair of timestamps, for beginning and end. The following rudimentary analysis will outline the predicates and the operations that need to be defined for handling temporal data.

There is only need for one boolean predicate for operation between timestamps, e.g., *ts1* and *ts2*, namely:

$$\text{EARLIER}(ts1, ts2) \;:=\; ts1 <= ts2$$

One can also define a couple of functions for conveniently selecting the first or the last timestamp from a set:

$$\text{EARLIEST}(ts1, ts2, \ldots) \;:=\; \text{MIN}(ts1, ts2, \ldots)$$

$$\text{LATEST}(ts1, ts2, \ldots) \;:=\; \text{MAX}(ts1, ts2, \ldots)$$

Once time intervals, e.g. *tr1*, are considered, there is need to define a predicate for the relationship of a timestamp to a time interval:

INSIDE(*ts1*, *tr1*) := EARLIER(*tr1.start*, *ts1*) AND EARLIER(*ts1*, *tr1.end*)

A corresponding predicate is also needed to determine whether a time interval is completely contained in another:

INSIDE(*tr1*, *tr2*) := EARLIER(*tr2.start*, *tr1.start*) AND EARLIER(*tr1.end*, *tr2.end*)

The next step is to define functions like the union of time intervals, which results in a new interval *u*, defined as:

u.start = LATEST(*tr1.start*, *tr2.start*) and
u.end = EARLIEST(*tr1.end*, *tr2.end*)

However, the above definition is correct if and only if:

INSIDE(*tr1.start*, *tr2*) OR INSIDE(*tr2.start*, *tr1*)

Otherwise, the two time intervals are not overlapping, and therefore their union is either the empty set or two disjoint time intervals, according to the preferred definition.

In the case of commonly used database systems, the temporal data are simply stored as an underlying basic type. The most frequently used approach is the use of `long int`s for timestamps and a pair of such types for time intervals. In these cases, all the above mentioned predicates and operations have to be implemented by the user, and performed, rather inefficiently, as arithmetic operations on integers. Some database systems have the internal type of timestamps for storing the information, but they do not usually provide any operations. Therefore, the implementation of the temporal structure in conventional database systems is usually lacking in efficiency.

3.10 Notes on system development

It should be noted that almost all our design and implementation decisions, while they were appropriate for our environment, do not represent the correct solution for any similar problem. For each and every one of them, one should carefully judge the functional requirements and the technical constraints that are probably present, either explicitly stated or inherent in some other components.

Different needs would almost certainly lead to different solutions. For example, the need to accommodate a much higher volume of dial-up calls, should probably point to a solution which utilizes a multi-threaded server which has incorporated almost all the functionality of the system, since calling external programs might result in a system that were unacceptably slow.

3.11 Future plans

While the Cordial system is currently fully functional and operational, a number of enhancements is planned for the near future.

As aforementioned, the migration of all user data to an infrastructure based on LDAP is currently under way. It is expected that, once this migration is completed, the authentication subsystem of the Cordial system will be significantly simplified. The deployment of the LDAP infrastructure may also have beneficial impact on other areas of the system, since it might also be used for storing the policies applicable in different cases. Since the policies, as described above, are in fact completely described and implemented in small scripts, they can easily be stored as attributes on a special branch of the LDAP tree structure.

Another possible enhancement is the move towards using a database system like PostgreSQL, which allegedly provides some of the temporal functionality outlined above.

Last, but not least, the reporting functionality is planned to be extended to include the production of customized reports for special needs. All the accounting data will also be migrated towards our data warehousing and data mining facility, in order to be analyzed. The ultimate goal is to discover any possibly hidden noteworthy patterns within the data, which in turn could possibly give insight about facts not previously discerned.

4. Monitoring and Reporting Facility

Another example of a tool that was developed for customized use is a general facility for monitoring and reporting the status of various systems and services. This system, which is yet unnamed, was developed out of growing needs to provide accurate information concerning the status of various servers and services. In contrast to the Cordial system that was described above, this facility was developed in a piecemeal way, with different components being designed and implemented as the need arose. As a consequence, the resulting system is inherently less well designed but, on the other hand, much more modular. This section will present briefly its main evolutionary stages.

4.1 Stage 1: simplicity itself

The primary functionality that this facility was designed to provide was a simple way of monitoring the status of the servers that were being used on our facilities. Since the number of these servers is not prohibitively large, we could get the same information by, for example, sending ICMP ECHO_REQUEST packets to each of these hosts, just by executing `ping`. Among the needed improve-

ments over the series of simple `ping`s, were a continuous execution, so that the status would get periodically updated, and a way of reporting the results so that the system would be usable from any of our personal workstations.

4.2 Stage 2: the move to the web

While the earliest versions of this software were, in fact, developed in such a way that they were running natively on the different workstations and operating systems that we were using, it soon became apparent that the proper way to have this information readily available was to publish it on a web page. Since the page would be generated as a result of a program execution, this solution also dealt with the need for continuous update of the information. All that was needed was to arrange the program to execute again after specific time intervals, which could be accomplished in a variety of ways.

4.3 Stage 3: monitoring services

After this simple reporting was installed and functioning satisfactorily, a different need arose that *services*, instead of simply computing systems, be monitored. Since most of the services, such as e-mail, web, news, and so on, were network-oriented, it was relatively easy, instead of performing the equivalent of `ping`, to perform the equivalent of a client of these services. Since the functionality in this point was diversified, a number of small programs was developed, each performing the simple task of monitoring a single service. It should be noted that these programs were extremely simple, rarely performing more than an initial connection to the server. However, chances were that if, for example, a given program could connect to a POP listening service and get a correct initial response, the rest of the functionality of this service would probably be intact. This was based on our observation that, for the services we were most interested in, severe failures usually resulted in the server process terminating, which made the service further unavailable to everyone.

4.4 Stage 4: major extensions

Once a number of such simple monitoring tools for different network services were developed, we faced the need to monitor the status of things unrelated to networking. An example of such measurement is the disk usage for the mail spool disk partition on the mail server. I was at that point in time that two major changes in the monitoring facility were introduced. The first involved the ability to check a metric by executing a program remotely, on another computer system. The second one was the realization that the results of the programs performing the monitoring need not be boolean, and the corresponding extension of the underlying mechanism to allow for arbitrary data to be communicated.

4.5 Current state

In its current state, the system is a collection of programs, each geared towards monitoring and reporting the status of a single service. The vast majority of these programs are written in Tcl and are therefore of rather small size. Depending on the desired functionality, a number of Tcl extensions are also used, such as `Tnm` for networking functions and `expect` for handling interactive programs like `telnet`, as well as libraries, such as the `cgi.tcl` which generates the web pages.

The diversity of the programs is hidden behind a unified interface, which is solely web-based and can therefore be used from any system connected on the network, even outside our offices. The security implications are minimal, since the system can only perform monitoring and reporting operations. However, as an added measure, the web server hosting the system is implementing rudimentary security checks.

Besides the generation of web pages, the system also supports mailing the relevant information to a predefined set of recipients, so that critical events can be dealt with in a timely matter.

4.6 Looking back

In retrospect, the evolution of this software seems extremely simple, and the modifications taken in each step seem almost trivial. However, the final result is a collection of software that implements all the required functionality, and, what might be more important, is simple and well-understood by the members of our staff. The resulting software is completely adapted to our environment, which has the advantage of being ideal for us, but also presents the disadvantage that it might not be the complete solution for a different environment.

The resulting software is undoubtedly not unique. Software which addresses the same needs, and which subsequently provides similar functionality, exists in various forms. Two well-known examples of widely used tools are the commercial HP OpenView suite and the freely available Big Brother system.

However, our experience has shown that the introduction and adoption of tools should always be performed with great caution. This is obviously more true in the case of massive environments like OpenView, but it is also relevant in the case of the simpler solutions like Big Brother. The most scarce commodity is the time of the people involved in the operation and maintenance of these tools, and any introduction of new environments must be followed by the appropriate training and familiarization. By developing custom tools, simple in both design and implementation, and gradually proceeding towards an integrated environment, the adoption path is much smoother.

5. Conclusions

During the development and deployment of system and network administration tools like the ones described above, a number of observations about the current state of such tools were made.

The most obvious observation is that system and network administrators need a lot of tools in order to cope with the difficulties they regularly face. The correct tools, if properly learned, can save a tremendous amount of time and help them accomplish tasks which would otherwise be impossible.

Tools like the ones mentioned above are generally available in a confusing variety of forms, with functionalities ranging from minimal to probably more than will ever be used. To be able to choose correctly, the administrators should ideally try as many as possible, in order to find the ones that closely match their requirements, their environment and can be adopted in the most straightforward manner.

Commercial offerings tend to come in the form of all-encompassing, massive systems, and therefore require a large and probably unavailable amount of resources, both of human time and of hardware.

On the other hand, the recent proliferation of freely available alternatives has created a situation where the choice between them is becoming continuously more difficult.

The specific components of each network environment concerning systems and services offered are different. While such differences might not be important for some of the more mundane system administration tasks, they are crucial for correctly designing, developing and deploying network administration tools.

The majority of generally useful tools comes in the form not of complete software systems, but in the form of libraries that can be tailored to a specific environment.

Finally, one should never underestimate the tremendous advantage offered by custom-made solutions. While the final result would probably be inferior to a large number of already existing systems, and its completion would probably continuously shift into the future, its focused functionality and the familiarity of the administrators with it might make it much more useful than any alternative.