FFHS
Fernfachhochschule Schweiz
Zürich | Basel | Bern | Brig
Mitglied der SUPSI

Bachelor-Thesis
Studiengang Informatik

Hochverfügbare, autonome Überwachung verteilter Systeme

Lukas Zurschmiede
BSc INF 2006 ZH2
06-655-138

Lommis, 09. Februar 2011

Betreut durch
Herr Dr. Frank Moehle,
cerrom engineering gmbh

Zusammenfassung

Aktuell geht die Tendenz in der Informatik, und vor allem im Bereich Software, immer mehr in Richtung *Cloud* sowie *Software as a Service*. Die Verfügbarkeit der Server und Dienste ist also essentiell wichtig, nicht nur für Business kritische Abläufe sondern auch immer mehr für private Personen. Dasselbe gilt auch für EMail- und Webserver sowie andere Dienste die für das Internet „lebenswichtig“ sind.

Produkte für die Überwachung solcher verteilter Systeme gibt es einige. Diese arbeiten jedoch meistens nach dem Prinzip: „*Ein zentraler Server überwacht die (de)zentralen Systeme*“.

Die **Hochverfügbare, autonome Überwachung verteilter Systeme** verzichtet komplett auf eine zentrale Instanz. Die einzelnen Systeme überwachen sich gegenseitig und handeln untereinander aus, wer wen überwacht. Dadurch entsteht ein Netzwerk, welches den Ausfall von einem oder mehreren Knoten verzeiht, denn ein System wird nicht nur von einem Rechner, sondern von mehreren anderen überwacht.

Abstract

Outsorcing software, or just some parts of it, into the *Could* to use it as *Software as a Service* is a general tendency in IT. The aviability of the servers and the services is extremely important not only for business critical processes also for private people. The same importance takes affect for web-, email- and other services and components which are important to the internet.

There exists some products to monitor such servers and services, but they are working mostly the same way: „*A central instance observes all other servers and services*“.

The **High available, self-gouverning monitoring of distributed systems** does not need a central instance at all. Every system monitors each other and negotiate automatic with themeselves who will be responsible for observing which other system. This creates a network that permits the failure of one or more nodes because they are monitored by more than only one other.

Danksagung

Besten Dank an:

delight software gmbh - http://www.delight.ch/
für die Unterstützung und die zur Verfügung gestellte Zeit, Infrastruktur und Systeme.

Herr Dr. Frank Moehle, cerrom engineering gmbh - http://www.cerrom.com/
der mich kompetent als Betreuer und mit wertvollen Ideen unterstützte.

Herr Christian Gavesi
für gute Disskusionen, aus welchen ich hilfreiche Informationen ziehen konnte.

Renate Zurschmiede, Sara Gavesi, Andrea Heller, Elias Zurschmiede, Christian Gavesi
für die Korrekturlesungen und die grammatikalischen und technischen Korrekturvorschläge.

Sowie meinem Hund Rantaplan,
für seine Geduld und Treue während der letzten Monate.

Inhaltsverzeichnis

1 Einleitung

Die Idee zu einem autonomen Überwachungssystem wurde schon vor einigen Jahren geboren. Die damaligen Systeme, wie auch die aktuellen, basieren alle auf dem Prinzip der zentralisierten Überwachung: Eine zentrale Instanz überwacht alle anderen Systeme und reagiert entsprechend bei einem Ausfall.

Die Problematik, welche sich bei uns in der Firma delight software gmbh immer stellte ist, dass wir viele kleine verteilte Server und Netzwerke überwachen sollten, welche meistens noch durch eine Firewall abgesichert sind. Hinzu kommt, dass die privaten Internetleitungen als nicht sehr stabil und zuverlässig angesehen werden konnten. Es kam also immer wieder zu totalen Überwachungs-Ausfällen und entsprechenden Fehlalarmen. Basierend auf dieser Tatsache wurde die Idee geboren, dass alle diese Systeme sich doch prinzipiell gegenseitig überwachen könnten. Zusätzlich soll das System sich automatisch auf neue Gegebenheiten einstellen können, ohne eine aufwändige Konfiguration.

Grosse Firmen, welche meistens zwei oder mehrere unterschiedliche Rechenzentren haben, setzen vielfach auf Überwachungssysteme, welche als Cluster betrieben werden können. Fällt ein Rechenzentrum oder System aus, wird automatisch ein anderes dafür eingesetzt. Dabei bemerken die Benutzer vielfach nicht mal, dass ein solcher Wechsel statgefunden hat, da alle Daten redundant auf allen Systemen verteil sind. Für solche interne abgeschirmte Netzwerke sind diese clusterfähigen Überwachungslösungen ideal; Nicht aber wenn viele unterschiedliche Systeme und Komponenten aus verschiedenen Netzwerken mit unterschiedlichen nicht redundanten Anbindungen überwacht werden sollen.

Da in einer kleinen Firma für solche „ideologischen“ Projekte meist nur begrenzt Zeit zur verfügung steht, wurde die Idee solange auf Eis gelegt, bis Zeit dafür aufgewendet werden konnte. Die Bachelor-Arbeit hat sich geradezu dafür angeboten, denn nicht nur eine Analyse der aktuellen Systeme, sondern auch eine Machbarkeits-Studie sind für ein solches Projekt unumgänglich.

In einem ersten Schritt werden bestehende Überwachungs-Systeme und deren Funktionsumfang analysiert. Anschliessend wird ein hochverfügbares autonomes Überwachungssystem theoretisch aufgebaut und hergeleitet, sowie im dritten Teil anschliessend

als Prototyp/TechDemo ausgearbeitet. Diese Technologie-Demonstration wird einen relativ eingeschränkten Funktionsumfang haben. Es soll damit gezeigt werden, dass und wie ein solches System funktionieren kann.

2 Bestehende Systeme und Methoden

Nachfolgend werden einige bestehende und etablierte Monitoring Systeme betrachtet und kurz zusammengefasst. Die Analyse soll deren Stärken wie auch Schwächen aufzeigen und dazu dienen, mögliche Funktionen und Techniken für das zu entwickelnde System ausfindig zu machen.

Ein weiterer Schwerpunkt der Analyse ist es, zu schauen, inwieweit das jeweilige System sich als hochverfügbare Monitoring-Instanz nutzen lässt. Dies setzt voraus, dass es nicht nur eine einzelne Instanz gibt, welche die überwachung ausführt, sondern mehrere. So kann gewährleistet werden, dass wenn eine Instanz ausfällt, die Überwachung dennoch statfindet. Ein System darf zudem nicht von zu vielen Umsystemen abhängig sein, denn je mehr Abhängigkeiten ein Produkt mit sich bringt, desto mehr Fehlerquellen und somit Fehlermeldungen und nicht überwachte Systeme kann es geben.

2.1 Systeme im Überblick

2.1.1 Cacti

Cacti[cac10] ist ein Monitoring-Tool basierend auf dem RRDtool[1] von Tobias Oetiker. Frontend und Backend sind in PHP[2] geschrieben, wobei die Konfiguration in einer MySQL-Datenbank[3] gespeichert wird. Zur Konfiguration der Endgeräte und der Graphen stehen viele Templates zur Verfügung, welche individuell angepasst werden können.

Damit man auch Kunden ihre eigenen überwachten Systeme anzeigen kann, besitzt Cacti die Möglichkeit Benutzer und Gruppen anzulegen sowie diese auf verschiedene Bereiche und Graphen einzuschränken.

Nebst RRDtool setzt Cacti einen funktionsfähigen Webserver mit PHP inklusive net-snmp sowie eine MySQL-Datenbank voraus. Sind diese Voraussetzungen gegeben, kann Cacti direkt in ein Verzeichnis auf dem Webserver kopiert und anschliessend über eine

[1] RRDtool[Oet10a] - http://oss.oetiker.ch/rrdtool/ - ist ein Quasi-Standard in der OpenSource Gemeinde für high performance Logging und Visualisierung. Vorteil des RRD-Formates ist es, dass die Datenbank grundsätzlich nicht überlaufen kann.

[2] http://www.php.net/

[3] http://www.mysql.com/

Weboberfläche konfiguriert werden. Für die regelmässige Prüfung aller konfigurierten Systeme muss nach der Konfiguration noch ein Cron-Job[4] eingerichtet werden, welcher alle 5 Minuten eine PHP-Datei ausführt.

Zusätzliche zu überwachende Systeme können mittels der Weboberfläche eingerichtet und in einer Baumstruktur abgelegt werden. Zur einfacheren Konfiguration stehen diverse Templates für Devices, Betriebssysteme und Graphen zur Verfügung. Diese Templates können angepasst und individualisiert werden. Dies ist für einen geübten Administrator zwar von Vorteil, jedoch kann die ganze Angelegenheit auch schnell sehr komplex und unübersichtlich werden.

Die Graphen werden dem Benutzer in einer Baumstruktur angezeigt. Diese weisen (je nach verwendetem Daten- und Graphtemplate) noch zusätzliche Informationen wie Minimal-, Maximal-, Durchschnitts- oder andere Werte auf. Einzelne Bereiche können mit der Maus markiert und dann genauer betrachtet werden. Dadurch können auch ältere Ereignisse im Nachhinein genauer analysiert werden.

Cacti bietet keine automatische Alerting-Funktionalität, diese muss über Plugins wie `monitor`[5] von `http://cactiusers.org/` nachträglich installiert werden.

Fazit

Das Haupt-Einsatzgebiet von Cacti liegt in der Überwachung von SNMP-fähigen Geräten. Dies können sowohl Serversysteme sein, bei welchen Ressourcen überwacht werden sollen, aber auch Router und andere Geräte die Trafficdaten per SNMP anbieten. Grundsätzlich kann Cacti sehr gut auf einem bestehenden Webserver installiert werden, denn die regelmässigen Prüfungen sind nicht sehr Performance lastig (je nach dem wie viele Systeme überprüft werden).

Durch das fehlende Alerting ist Cacti weniger geeignet für hochverfügbare Systeme, denn die Graphen müssen visuell überwacht werden. Cacti hat seine Stärken in der Verwaltung und Darstellung der Daten sowie der Möglichkeiten zur Datenabfrage und -Aufbereitung (individuelle SNMP-OIDs, etc.).

2.1.2 SmokePing

SmokePing[Oet10b] ist ein auf dem RRDtool[Oet10a] basierendes Tool zur Überwachung der Netzwerk-Latenzen. Wie das RRDtool ist auch SmokePing von Tobi Oetiker entwickelt. Das komplette Front- und Backend von SmokePing ist in Perl geschrieben und

[4] Unter Windows ein *Geplanter Task*

[5] Das Monitor-Plugin zeigt alle Hosts in einem Raster an. Fällt ein Host aus, wird er hervorgehoben und ein Alarm-Sound erklingt.

setzt neben einem Webserver[6] und einigen Perl-Modulen[7] nur FPing[Dzu10] voraus. Neben Latenz-Tests können mit SmokePing auch Performance-[8], DNS-[9], SSH-[10], Radius-[11], LDAP-[12] und andere Tests durchgeführt werden.

SmokePing muss, wenn alle Voraussetzungen erfüllt sind, lediglich in ein Verzeichnis auf dem Server kopiert werden. Dieses Verzeichnis muss anschliessend im Webserver als CGI-Verzeichnis konfiguriert werden. Für die Überwachung muss der SmokePing-Daemon noch so eingerichtet werden, dass dieser auch nach einem Neustart wieder automatisch gestartet wird. Damit der SmokePing-Daemon die Daten richtig speichert und die korrekten Dateien erzeugt, müssen zwei Perl-Dateien angepasst und überprüft werden.

SmokePing wird grundsätzlich über eine einzelne Datei konfiguriert. Dabei werden die zu überwachenden Systeme mittels einer speziellen Notation in einer Baumstruktur angelegt und parametriert. Jedes System kann mit unterschiedlichen Überwachungen und Alerting-Regeln belegt werden.

Die Graphen werden in der durch die Konfiguration festgelegten Baumstruktur dargestellt. Die einzelnen Ästen enthalten die jeweiligen Übersichtsgraphen, während die Blätter alle konfigurierten Graphen mit den verschiedenen Zeitintervallen zeigen. Zur nachträglichen Prüfung von Ereignissen können Bereiche in einem Graphen markiert und vergrössert dargestellt werden.

Fällt ein System aus, wird das Alerting von SmokePing in Betrieb gesetzt. Dieses kann pro überwachtem System durch mehrere Alerting-Mechanismen individuell gestaltet werden. So kann zum Beispiel definiert werden, dass bei einem Packet-Lost von 50% über einen gewissen Zeitraum der erste Alert gesendet werden soll. Fällt das System komplett aus, soll jedoch ein anderer Alert gesendet werden. Aktuell kennt SmokePing drei verschiedene Alert-Typen: rtt, loss und matcher[13]. Die Alerts werden nicht durch SmokePing direkt versendet - für die Übermittlung dieser müssen externe Applikationen oder Skripte vorhanden sein.

[6] Der Webserver muss CGI fähig sein. Präferiert wird ein Apache mit der Option SuExec, welche es erlaubt CGI-Skripte als normalen Benutzer auszuführen.
[7] Perl-Module für SmokePing: LWP::UserAgent, CGI::Carp
[8] Für Performance-Tests muss EchoPing vorhanden sein: `http://www.echoping.sf.net/`
[9] Für DNS-Tests muss das Perl Modul Net::DNS sowie das Kommando `dig` vorhanden sein: `http://www.isc.org/bind/`
[10] Für SSH-Tests muss das Perl Modul IO::Soket::SSL sowie OpenSSH vorhanden sein: `http://www.openssh.org/`
[11] Für Radius-Tests muss das Perl Modul Authen::Radius vorhanden sein
[12] Für LDAP-Tests muss das Perl Modul Net::LDAP vorhanden sein
[13] Matcher Alerts sind Plugins, sie erweitern die Alert-Konditionen. Bekannte Matcher sind z.B. AvgRatio, CheckLatency, CheckLoss, Median, MedRatio

Fazit

SmokePing findet seinen Einsatz vor allem in der Überwachung von Netzwerk-Latenzen sowie Latenzen bei verschiedensten Diensten. Durch die geringe Systemlast kann SmokePing sehr gut als nebenläufiges System auf einem existierenden Server eingerichtet werden. Durch das ausgeklügelte Alerting System kann SmokePing zudem auch als Hintergrundprozess laufen und muss nicht dauernd manuell überwacht werden.

2.1.3 Zenoss Core

Zenoss Core[zen10] stellt ein umfangreiches Grundpaket für Systemmonitoring und Systemmanagement zur Verfügung. Mittels Zenoss Core kann das komplette Netzwerk nachmodelliert und überwacht werden. Es bietet Module zur Überwachung der Erreichbarkeit[14] und Performance[15] von Systemen an. Neben einer freien Community-Lösung kann Zennoss Core auch als Enterprise-Version erworben werden, welche zusätzliche Funktionen[16] sowie Support beinhaltet.

Die Voraussetzungen für eine Zenoss-Installation variieren. Basisvoraussetzung ist jedoch *MySQL*, *net-snmp*, *GCJ* und *OpenMP*. Für eine manuelle Installation unter einem Linux/Unix-System werden *GCC/G++*, *GNU build environment*, *SWIG* und *Autoconv* vorausgesetzt. Zusätzlich müssen manuell Verzeichnisse und ein Systembenutzer angelegt, sowie ein Init-Skript erstellt werden, welches die Zenoss-Dienste bei einem Systemstart ausführt. Bei der Installation über einen Paketmanager entfallen alle diese Schritte und Systemeingriffe.

Ist das System installiert und gestartet, kann Zenoss Core direkt über den Browser aufgerufen und konfiguriert werden. Die verschiedenen Systeme und Komponenten können dabei mittels Plugins, Kommandos, SNMP-OIDs, etc. konfiguriert werden. Für weiterführende Überwachungen steht zudem die Möglichkeit bereit, ein vorkonfiguriertes Kommando mittels SSH[17] auf einem entfernten System auszuführen sowie die SNMP-OIDs zu erweitern[18].

Durch verschiedene Benutzergruppen und Berechtigungen kann das System auch für Kunden geöffnet werden, welche zum Beispiel nur ihre eigenen Komponenten einsehen

[14] ICMP, SNMP, TCP/IP-Dienste, Windows-Services und Prozesse, Linux/Unix-Prozesse, Nagios-Plugins, ...

[15] JMX-Performance von J2EE-Servern, Nagios- und Cacti-Collection Scripts, ...

[16] Virtualisation-, Cloud-, Cisco UCS-Monitoring und Management

[17] Für eine SSH-Verbindung muss sich der Benutzer unter welchem Zenoss Core läuft mittels Zertifikat (ohne Passwort) auf dem entfernten System anmelden können

[18] Für individualisierte SNMP-OIDs muss sowohl der Überwachungs-Server als auch der entfernte Server angepasst und mit den zusätzlichen Informationen ausgestattet werden, was schnell zu einer komplexen Angelegenheit werden kann

oder auch konfigurieren können. Jeder Benutzer, der sich anmeldet, sieht zuerst ein Dashboard, welches die wichtigsten Ereignisse und Alerts aufzeigt. Verschiedene Unterseiten zeigen weitere Details zu den verschiedenen Systemen und Meldungen. Diese sind jedoch für Laien teilweise nur schwer verständlich und erscheinen einem ungeübten Benutzer schnell kryptisch und komplex.

Das Alerting bei Zenoss Core basiert auf Regeln, welche sehr umfangreich konfiguriert und verschachtelt werden können. So können ähnlich wie bei SmokePing (siehe Kapitel 2.1.2) verschiedene Schweregrade definiert und nacheinander ausgelöst werden. Das Alerting bei Zenoss Core basiert ausschliesslich auf SMTP, was bei einem EMail-Serverausfall, respektive dessen „nicht-Erreichbarkeit“, schwerwiegende Probleme nach sich ziehen kann - Systemausfälle können nicht mehr gemeldet werden, die Fehler werden also wahrscheinlich nicht bemerkt und nicht behoben.

Fazit

Zennoss Core wird wohl hauptsächlich in der Server- und Dienstüberwachung eingesetzt. Durch das Alerting und dessen Konfigurationsmöglichkeiten kann das System gut ohne visuelle Überwachung eingesetzt werden.

2.1.4 Zabbix

Zabbix[zab10] ist eine OpenSource Monitoring Lösung, welche sowohl Polling[19] wie auch Trapping[20] zur Überwachung von Netzwerken und Systemen anbietet. Durch zusätzliche, auf den Systemen installierte Agenten, können neben den standardmässig angebotenen Informationen in SNMP-Anfragen noch weitere Daten abgefragt werden. Die Überwachung von TCP/IP-Diensten und ICMP-Prüfungen gehören ebenfalls zu den Grundfunktionalitäten von Zabbix. Durch die `Auto-Discovery`-Funktion[21] kann ein Netzwerk schnell und einfach in Zabbix konfiguriert und ohne grossen Aufwand überwacht werden. Zusätzlich zur zentralen Weboberfläche wird auch eine API angeboten, über welche Informationen ausgelesen werden können.

Zabbix ist unterteilt in vier Hauptkomponenten. Der Zabbix-Server stellt die zentrale Stelle dar, welche alle Aktionen und Events sendet, empfängt, auswertet und Alerts auslöst. Der Zabbix-Proxy ist ein Meldungs-Puffer, welcher Nachrichten empfängt und dem

[19] Polling bedeutet das ein Status explizit angefragt wird, zum Beispiel ICMP/Echo-Ping
[20] Trapping heisst, dass Meldungen empfangen werden, welche nicht angefragt worden sind, zum Beispiel SNMP-Traps
[21] Durch Auto-Discovery wird das gesamte Netzwerk automatisch ausgelesen und die verfügbaren Geräte erkannt

zentralen Zabbix-Server weiterleitet. Die Zabbix-Agents werden auf Systemen installiert, um lokale Ereignisse und Ressourcen zu überwachen und den zentralen Server darüber zu informieren. Das Web-Frontend gewährt visuellen Zugang zu allen Daten, Graphen und der Konfiguration.

Das Zabbix Frontend ist in PHP geschrieben und setzt daher einen Webserver mit PHP[22] voraus. Zusätzlich sollte der Server OpenIPMI- und SSH-Unterstützung bieten, damit der komplette Funktionsumfang genutzt werden kann. Bei einer manuellen Installation der Zabbix-Komponenten werden zusätzliche Bibliotheken[23] vorausgesetzt, welche bei der Installation mittels einem Paket-Manager oder unter Windows nicht gebraucht werden. Nach der Installation müssen noch Services und Startskripte eingerichtet werden um dem Server, den Agenten oder den Proxy automatisch auszuführen. Die Weboberfläche muss in ein Verzeichnis auf dem Webserver kopiert und mittels einem Browser anschliessend konfiguriert werden.

Die Grundkonfiguration aller Zabbix-Dienste (Server, Proxy und Agent) wird über eine Konfigurationsdatei gemacht. Die weiterführenden Einstellungen werden anschliessend über die zentrale Weboberfläche getätigt. Durch vorgefertigte Host-Templates lässt sich schnell ein einfaches Monitoring einrichten. Zabbix erlaubt jedoch auch, jeden Event und jede Aktion mit Makros und individuellen Prüfungen zu untersuchen oder neue Datenquellen und Events anzulegen. Dies erlaubt es, ein hoch komplexes und ausgereiftes Monitoring-Netzwerk zu erstellen, welches jedoch den Nachteil hat, dass es oft komplex und nur noch schwer konfigurierbar wird.

Das Alerting von Zabbix ist ähnlich wie das Monitoring: Einfach in der Grundausstattung und sehr stark individualisierbar. Events werden empfangen und entsprechende Aktionen eingeleitet. Die verschiedenen Aktionen können die Events dann entweder zurückhalten oder an die nächsten Aktionen weiterleiten. So kann, wie zum Beispiel bei SmokePing (siehe Kapitel 2.1.2), ein Event unterschiedlich abgehandelt werden und je nach Ereignis und Zeitpunkt, ein unterschiedlicher Alert versendet werden. Ein Alert kann über SMTP, Jabber, GSM-Module oder individuelle Skripte versendet werden.

Fazit

Zabbix wird vielfach in grösseren und auch verteilten Netzwerken eingesetzt. Das Einsatzgebiet ist dabei auf Grund der Erweiterbarkeit und der Flexibilität nicht beschränkt.

[22] PHP muss GD, TrueType, BCMath, XML, Session, Socket, MultiByte und MySQL/Oracle/PostgreSQL/SqLite3 unterstützen

[23] Build-Utils, GCC/G++ sowie Header und Libraries von MySQL (resp. diejenigen von Oracle, PostgreSQL, SQLite), net-snmp, Iksemel (für Jabber-Alerts) und Libcurl

Durch das Alerting und dessen Konfigurationsmöglichkeiten kann ein System gut ohne visuelle Überwachung eingesetzt werden. Die hochverfügbare Überwachung scheitert auch bei Zabbix aufgrund des zentralen Überwachungsservers, auch wenn andere Gegebenheiten wie die Agenten darauf ausgelegt zu sein scheinen.

2.1.5 Nagios Core

Nagios Core[nag10] stellt eine zentrale Einheit zur allumfassenden Überwachung von IT-Infrastrukturen bereit. Dabei können sowohl Systeme und Applikationen wie auch Dienste mittels Polling und Trapping überwacht werden. Durch die NRPE-Erweiterung[24] können auf den zu überwachenden Systemen zusätzliche Informationen abgefragt werden. Die Fähigkeit, Daten über Agenten zu empfangen, wird von Nagios und vielen anderen verkauft als „Clustering“ - was es jedoch nicht ist, denn die Agenten sammeln nur Daten und reichen diese an die zentrale Instanz weiter. Durch die grosse Verbreitung und Akzeptanz von Nagios sind viele Erweiterungen, Plugins, Patches und Tutorials im Internet vorhanden. Eine Sammlung mit über 350 Addons, 1700 Plugins und vielem mehr, ist beispielsweise unter `http://exchange.nagios.org/` zu finden.

Nagios setzt nicht wie andere Monitoring-Systeme auf eine Skript-Sprache, sondern ist komplett in C geschrieben. Für die zentrale Web-Oberfläche wird jedoch ein Webserver vorausgesetzt, welcher für die Erstellung der Graphen die GD2-Bibliothek benötigt. Die Installation ist recht umfassend und setzt einiges an Wissen auf dem jeweiligen Betriebssystem voraus. Zuerst müssen Benutzer und Gruppen angelegt werden und anschliessend der Code, mittels der Angabe des Web-Verzeichnisses, konfiguriert und mittels fünf verschiedenen Kommandos[25] kompiliert und installiert werden. Anschliessend müssen alle Beispiel-Konfigurationsdateien kopiert und auf das System angepasst werden. Sind alle Konfigurationen gemacht, muss noch die Web-Oberfläche konfiguriert[26] und mittels einem Passwort[27] abgesichert sowie der Webserver neu gestartet werden. Sollen noch Plugins in Nagios eingebettet werden, müssen diese ebenfalls zuerst heruntergeladen, entpackt, kompiliert und installiert werden. Schlussendlich können die Start-Skripte kopiert, angepasst und dann Nagios Core gestartet werden.

Die gesamte Konfiguration von Nagios und den zu überwachenden Systemen geschieht über Konfigurationsdateien - eine grafische Konfigurationsmöglichkeit via Browser wird standardmässig nicht angeboten. Die Konfiguration kann entweder in einer zentralen

[24] Die NRPE-Erweiterung ist ein Agent auf dem zu überwachenden System, welche direkt vom Nagios Core Server angesprochen werden kann

[25] `make all; make install; make install-init; make install-config; make install-commandmode`

[26] `make install-webconf`

[27] `htpasswd -c /path/to/nagios/etc/htpasswd.users nagiosadmin`

Datei erfolgen oder über verschiedene Dateien, welche jedoch manuell eingebunden werden müssen. Bevor ein System überwacht werden kann, muss eine Host-Konfiguration[28] angelegt werden. Dienste, welche auf einem Host überwacht werden sollen, können anschliessend über einen `service`[29]-Abschnitt definiert werden. Diese Service-Abschnitte basieren auf Kommandos, welche mittels `command`-Blöcken[30] definiert werden können. Nach jeder Anpassung einer Konfiguration sollte zuerst die Konfiguration durch Nagios geprüft[31] und anschliessend der Serverdienst neu gestartet werden.

Fazit

Nagios wird vielfach in grösseren und auch verteilten Netzwerken eingesetzt. Das Einsatzgebiet ist dabei aufgrund der Erweiterbarkeit und der Flexibilität sehr gross. Durch das Alerting und dessen Konfigurationsmöglichkeiten kann das System gut ohne visuelle Überwachung eingesetzt werden. Die hochverfügbare Überwachung scheitert auch bei Nagios aufgrund des zentralen Überwachungsservers, auch wenn andere Gegebenheiten wie die Agenten darauf ausgelegt sind. Ein zusätzlicher Schwachpunkt ist die Konfiguration, denn diese kann durch die verschiedenen Abschnitte und Dateien sehr komplex und unübersichtlich werden.

2.1.6 WhatsUp gold

WhatsUp Gold[wha10] ist eine allumfassende Netzwerk- und Systemüberwachung, welche sich durch eine einfache Bedienung und Konfiguration auszeichnet. WhatsUp Gold ist ausschliesslich unter Windows mit IIS[32] lauffähig und setzt einen Microsoft SQL[33]-Server voraus. Weitere Voraussetzungen sind Microsoft Internet Explorer ab Version 7, Microsoft .NET Framework ab 3.51 SP1, Windows Scripting Host ab v5.7 und Microsoft SAPI ab 5.1 für Text-to-Speech Aktionen.

Wenn alle Voraussetzungen gegeben sind, wird wie unter Windows üblich WhatsUp Gold mittels einem Installer installiert. Die anschliessende, initiale Konfiguration kann über einen Setup-Assistenten gemacht werden. Dabei können die grundlegenden Alert-Zeiten und Funktionen definiert sowie Systeme hinzugefügt werden. Die Geräte können

[28] define host { ... } - Definition eines Hosts basierend auf einem Host-Template
[29] define service { ... } - Definition eines Service zur Überwachung eines command
[30] define command { ... } - Definition eines Kommandos zur Überwachung von SNMP, NPRE, etc.
[31] Die Prüfung der Konfiguration geschieht durch das manuelle aufrufen des Nagios-Binaries mit dem Parameter `-v /path/to/nagios/etc/nagios.cfg`
[32] WhatsUp Gold setzt einen IIS-6 oder IIS-7 voraus
[33] Bei den MS-SQL Servern ist zu beachten, dass auch die Gratisversionen zum Einsatz kommen können, diese jedoch ein Datenlimit haben

dabei durch ein HostDiscovery automatisch gesucht oder manuell hinzugefügt werden. Den Geräten können anschliessend Rollen[34] vergeben werden. Durch die automatische Suche nach Geräten wird ebenfalls automatisch ein Netzwerkplan erstellt.

Nach der Konfiguration kann der Status des Netzwerkes und der Systeme über die zentrale Weboberfläche betrachtet werden. Diese kann teilweise individualisiert werden, damit die wichtigsten Systeme jederzeit im Auge behalten werden können. Durch Anwählen eines Hosts können weitere Details sowie vergangene und aktuelle Alerts und Events eingesehen werden.

Das Alerting bei WhatsUp Gold folgt einfachen Regeln, welche global konfiguriert werden können. Die Regeln beschreiben, was nach welcher Zeit gemacht werden soll. So kann zum Beispiel eingestellt werden, dass beim Bemerken eines Ausfalls zuerst nichts unternommen, wenn der Ausfall über mehr als 5 Minuten andauert ein SMS versendet und wenn der Ausfall länger als 10 Minuten dauert zusätzlich noch ein EMail gesendet werden soll. Alerts können bei WhatsUp Gold nur über EMail oder auch über angehängte GSM-Module per SMS versendet werden.

Fazit

WhatsUp Gold ist ein sehr umfangreiches Paket, welches mit kostenpflichtigen Erweiterungen individuell auf den eigenen Betrieb ausgelegt werden kann. Durch den hohen Bedarf an Ressourcen sowie der Voraussetzung eines IIS inklusive MS-SQL sollte WhatsUp Gold, wenn möglich, auf einem eigenen Server installiert werden. Dem Marketing zufolge ist WhatsUp Gold ein hochverfügbares Monitoring System. Bei genauerer Betrachtung scheitert auch dieses System an einer einzelnen zentralen Überwachungs-Instanz, jedoch mit der Möglichkeit von FailOver-Destinationen[35].

2.1.7 NeDi

NeDi[Ric10] stellt eine Sammlung von Skripten dar, welche mittels CDP (Cisco Discovery Protocol) und LLDP (Link Layer Discovery Protocol) ein komplettes Netzwerk automatisch untersuchen. Geräte, welche SNMP unterstützen, werden mittels den SNMP-Location-Strings[36] automatisch lokalisiert, auf einem Netzwerkplan eingefügt und gekennzeichnet. Bei Geräten, welche keine SNMP-Unterstützung anbieten, wird versucht

[34] Rollen sind bei WhatsUp Gold Definitionen, welche den Gerätetyp beschreiben. Dies sind zum Beispiel: Webserver, Switch, Mailserver, Drucker, etc.

[35] Eine FailOver-Destination ist eine weiter Installation, welche bei einem Ausfall des Hauptsystems zum Einsatz kommt.

[36] Die SNMP-Location-Strings sollten immer den Aufbau `Region;Stadt;Gebäude;Stockwerk;Raum;[...]` haben

anhand der Informationen der CDP/LLDP-Anfragen zu erfahren, um was für ein Gerät es sich handelt. Die Erstellung des Netzwerkplans ist jedoch nicht abhängig von der SNMP-Fähigkeit. Dieser wird mittels den Daten aus den CDP/LLDP-Anfragen erstellt. Die gefundenen Systeme und Geräte werden in regelmässigen Abständen erneuert und können über SNMP, SSH oder einfache Telnet-Verbindungen weiter untersucht und überwacht werden. Die gesammelten Daten werden in einer zentralen MySQL-Datenbank gespeichert. Zusätzlich kann für Langzeit-Visualisierungen noch RRDtool mitverwendet werden. NeDi ist so konzipiert, dass die Überwachungs-Skripte, die Weboberfläche und der Datenbankserver auf getrennten Systemen laufen können.

NeDi ist komplett in Perl geschrieben und setzt nur wenige zusätzliche Module[37] voraus. Die Weboberfläche ist in PHP geschrieben und setzt demzufolge einen Webserver inklusive PHP mit Net-SNMP Unterstützung voraus. Durch die Verwendung von Skript-Sprachen kann NeDi einfach auf einen Server kopiert[38] und anschliessend konfiguriert werden. Die Datei `nedi.conf` stellt dabei die zentrale Konfiguration dar. Optional kann zudem eine Liste mit IP-Adressen angelegt werden[39], bei welchen NeDi mit der Suche nach neuen Geräten anfangen soll. Nachdem NeDi initialisiert[40] worden ist, kann das Netzwerk nach Geräten durchsucht werden[41]. Für das Monitoring müssen jeweils beim Systemstart noch die zwei Skripte `moni.pl` und `syslog.pl` sowie der SNMP-Trap Dienst `snmptrapd` gestartet werden. Damit NeDi das Netzwerk automatisch nach neuen oder nicht mehr vorhandenen Geräten durchsucht, sollte zudem noch ein Cron-Job[42] eingerichtet werden.

Neben dem Netzwerkplan, welcher eine der zentralen Einheiten von NeDi darstellt, können über die Weboberfläche alle Informationen von den Geräten angezeigt werden. Dabei werden je nach vorhandenen Daten, Graphen und Listen mit überwachten Diensten und deren Verfügbarkeit angezeigt. Über das Reporting können Daten von Netzwerken, Geräten, Modulen, Interfaces und vielem mehr live zusammengestellt und ausgewertet werden. Bei der Anzeige des Netzwerkplans kann gewählt werden, welche Bereiche/Gruppen dargestellt werden sollen. Die Gruppen werden dabei aus dem SNMP-

[37] Net::SNMP, Net::Telnet::Cisco, Algorithm::Diff, DBI, DBD::MySQL, LWP, Net::SSH::Perl, Net::SMTP inklusive libnet

[38] Wird NeDi nicht nach `/var/nedi/` installiert, müssen in einer PHP-Datei noch Pfade angepasst werden. Das Komplette `html`-Verzeichnis muss nach der Installation in ein Verzeichnis auf dem Webserver kopiert werden.

[39] Die Datei `seedlist` beinhaltet alle gefundenen IP-Adressen. Initial kann eine Liste eingegeben werden bei welchen NeDi anfangen soll mit dem Polling.

[40] Die Initialisation von NeDi und der Datenbank geschieht mittels `./nedi.pl -i`

[41] Das Netzwerk kann durch `./nedi.pl -cob` nach Geräten durchsucht werden

[42] Je nach Netzwerk Grösse muss darauf geachtet werden, dass nicht zwei Scan-Prozesse parallel aufgerufen werden.

Location-Strings extrahiert und zur Auswahl angeboten. Zusätzlich können noch weitere Filter definiert und Traffic- oder andere Graphen eingeblendet werden (sofern die Geräte diese Informationen anbieten).

Fazit

NeDi wird wohl vielfach zur Überwachung der Verfügbarkeit von Geräten in einem Firmennetzwerk eingesetzt. Zusätzlich zu der Verfügbarkeit können auch einfache Dienste zusätzlich in das Monitoring eingeschlossen werden. NeDi scheint zudem eher für eine manuelle/visuelle als für eine autonome Überwachung ausgelegt zu sein.

2.1.8 Fazit aller Systeme

Die meisten Monitoring-Systeme sind auf ein spezielles Gebiet in der Netzwerküberwachung ausgelegt. Sei dies nun die automatische Komponenten-Lokalisierung und Überwachung wie sie NeDi bietet oder die reine Dienst- und Performaceüberwachung von Cacti und SmokePing.

Interessante Funktionen bieten fast alle untersuchten Systeme: **AutoDiscovery** von NeDi und Zabbix, **Proxys und Agents auf entfernten Systeme** von Zabbix und Nagios, **Performance-Tests** von SmokePing, **individuelle SNMP-OIDs** bei Cacti und Zenoss sowie die Verwendung von verschiedenen Netzwerk-Technologien wie **SNMP(-Traps), Syslog, ICMP, OpenIPMI** etc.

Für die hochverfügbare Überwachung ist keines der Systeme ausgelegt. Alle Systeme setzten auf eine zentrale Instanz, welche alle umliegenden Systeme überwacht. Fällt diese zentrale Instanz aus, ist das komplette Netzwerk nicht mehr überwacht. WhatsUp Gold beschreitet diesbezüglich eine Vorreiterrolle, denn fällt hier die Hauptinstanz aus, übernimmt dessen Funktion eine FailOver-Destination - dieses „Clustering“ wird auch von einigen Spezial-Systemen angewendet, welche jedoch eher für Rechenzentren und spezielle Server denn für Individualüberwachungen sind. Zabbix und Nagios setzen diesbezüglich auf eine andere Technik, den Proxy. Dieser empfängt die Nachrichten von den umliegenden Agenten und Systemen und leitet sie dann an die zentrale Instanz weiter, sobald diese wieder verfügbar ist.

Keines der untersuchten Systeme kann ohne zusätzliche Technik, wie einer FailOver-Anbindung der zentralen Instanz oder des betreibens dieser in einem Cluster, hochverfügbar überwachen. Die meisten Systeme setzen zudem noch voraus, dass verschiedene Services wie ein Webserver, Datenbankserver, etc. funktionstüchtig sind. Es wird also genau das vorausgesetzt, was mit einer Überwachungs-Software eigentlich überwacht

werden sollte: Eine Lauffähige Infrastruktur und Server. Ist das nicht der Fall, laufen also gewisse Dienste nicht, wird das Netzwerk nicht überwacht und es kann nicht gemeldet oder visuell betrachtet werden, dass etwas nicht läuft.

Bei Firmen mit entsprechendem Budget mag Clustering oder der Einsatz von Hersteller-Abhängigen Lösungen kein Hindernis sein, kleinere Firmen, welche jedoch keinen entsprechenden Etat aufweisen, können sich den Einsatz dieser Technik nicht leisten. Dies ist Anlass genug, ein solches System zu planen und durch eine TechDemo dessen Funktionalität und Wirksamkeit zu belegen.

3 Theoretische Abhandlung

Die heutigen Netzwerke, vorallem von kleineren Firmen, basieren vielfach noch auf IPv4[1] mit einem stark begrenzten Adressraum[2]. Dies hat den Grund, dass die Internet Service Provider (ISP) ihre Angebote (oder oder Technik) vielfach noch nicht auf den IPv6-Standard aktualisiert haben, da sie noch über eine genügend grosse Anzahl an IPv4-Adressen verfügen[3]. Das hat zur Folge, dass Server welche nur innerhalb einer Firma genutzt werden, in einem privaten Netzwerk stehen und durch NAT[4]-Regeln abgeschirmt sind. Eine Firma hat somit meistens nur eine öffentliche IP-Adresse, kann durch eine NAT-Firewall aber verschiedene Dienste von unterschiedlichen internen Servern öffentlich anbieten. Es können aber auch Systeme im Netzwerk existieren, welche IP-Technisch komplett vom Internet abgeschirmt sind, da sie aus dem Internet nicht erreichbar sind. Ein Überwachungssystem soll aber nicht nur die eine öffentliche IP-Adresse prüfen, sondern auch in der Lage sein, alle weiteren nicht öffentlich zugänglichen Server und Dienste zu prüfen. Diese Problematik wird in Kapitel 3.1 behandelt. Auf die grundlegende Problematik von IPv4 und IPv6 basierenden Netzwerken sowie die verschiedenen NAT-Techniken und -Prinzipien wird in dieser Arbeit nicht eingegangen. Darüber existieren nicht nur viele Berichte im Internet sondern auch diverse Artikel in Fachzeitschriften und Büchern, wie Beispielsweise „Computernetzwerke von Andrew S. Tannenbaum [Tan03]“.

Ein weiterer Schwerpunkt liegt in der Definierung von Geräten in einem Netzwerk. Systeme sollten nicht nur manuell in einer Liste eingetragen, sondern auch automatisch aufgefunden werden können. Dies nicht nur im aktuellen, lokalen Netzwerk, sondern auch in einem entfernten, welches ebenfalls überwacht wird. Das Auto-Discovery, auch Host-Discovery genannt, muss nicht nur erkennen können, ob auf einem gefundenen

[1] IPv6 ist in den Startlöchern, kann jedoch noch nicht bei jedem Dienstleister genutzt werden.

[2] IPv4 hat einen Adressraum von 2^{32} Adressen, IPv6 hingegen 2^{128}. IPv6 ist also um den Faktor 2^{96} grösser und bietet viele Vereinfachungen auf Protokoll- und auf Sicherheitsebene.

[3] Dieser Fakt wird sich in den kommenden Monaten wahrscheinlich ändern, da seit dem 1. Februar 2011 die restlichen verfügbaren 8-er Blöcke auf die fünf regionalen Internet Registries (RIRs) verteilt worden sind: http://www.apnic.net/publications/news/2011/delegation. Es können somit nur noch wenige tausend IPv4-Adressen bei den den RIRs angefordert werden.

[4] Eine NAT basierende Firewall kann so konfiguriert werden, dass nur eine Anfrage an Port 80 an einen bestimmten Server innerhalb des Netzwerkes weitergeleitet wird, nicht jedoch eine Anfrage auf einem anderen Port.

System eine Instanz der Überwachungs-Software läuft, sondern auch, ob dies ein einfach zu überwachendes oder sogar ein zu ignorierendes Gerät darstellt.

Sind alle Geräte aus dem aktuellen und allen anderen Netzwerken bekannt, müssen diese anschliessend automatisch fragmentiert werden. Es muss also bestimmt werden, welche Systeme welche Komponenten und Dienste aus dem internen und aus entfernten Netzwerken überwachen, wo die Konfiguration gespeichert wird und wie diese aufgeteilt ist, was beim auffinden eines Problems passiert und wie dieses gemeldet werden kann, sowie was für Techniken es zur Systemüberwachung und Datenübermittlung gibt. Bei der Übermittlung von Problemen, also dem senden einer Alert-Meldung[5], muss besonders beachtet werden, dass ein und das selbe Problem nicht mehrmals von verschiedenen Überwachungsservern gesendet wird.

3.1 Problemfall NAT

Sollen Server innerhalb eines durch NAT abgeschirmten Netzwerkes überwacht werden, müsste für jeden Server eine NAT-Regel definiert werden, damit dieser nicht nur intern, sondern auch von extern erreichbar und somit überwachbar ist. Wenn nun mehrere Dienste, zum Beispiel HTTP[BLFF96][FGM+99], SMTP[Kle08], POP3[MR96], IMAP[Cri03], ICMP[Pos81] und FTP[PR85], auf einem Server überwacht werden sollen, müsste nicht nur eine NAT-Regel definiert werden sondern deren sechs. Je mehr Server und Dienste also in einem abgeschirmten Netzwerk von ausserhalb erreichbar sein sollen, desto mehr NAT-Regeln müssen erstellt werden. Die Anzahl kann durch

$$R = \sum_{H=1}^{M} S_H \tag{3.1}$$

berechnet werden, wobei R die Anzahl der NAT-Regeln, M die Anzahl Server, H der aktuelle Server und S_H die Anzahl Dienste auf diesem beschreiben.

Die Problematik bei den erstellten NAT-Regeln liegt nicht nur bei der begrenzten Anzahl an Ports[6] bei TCP[CDS74] und UDP[Pos80] basierenden Protokollen, sondern diese Regeln stellen ein erhebliches Sicherheitsrisiko dar. Alle Systeme welche Anfragen an die NAT-Firewall stellen können, in den meisten Szenarien wahrscheinlich das gesamte Internet, sind nun ebenfalls in der Lage die verschiedenen Dienste auf den internen Servern nutzen.

[5] Eine Alert-Meldung ist eine Alarmierungs-Nachricht.
[6] TCP/IP und UDP verfügen über jeweils 65535 Ports

3.1.1 Source basierende NAT-Regeln

Eine Möglichkeit dieses Problem zu beseitigen wäre, die NAT-Regeln nicht nur an den Port und das Protokoll zu binden, sondern auch auf den Source-Host[7]. Somit müsste nicht nur eine Regel pro Host und Dienst erstellt werden, sondern zusätzlich noch eine für jeden Source-Host in allen entfernten Netzwerken, welche ebenfalls in das Überwachungsnetzwerk eingeschlossen sind (siehe Abbildung 5.1 im Anhang). Die Anzahl der benötigten Regeln kann durch eine Multiplikation der Anzahl Dienste mit der Anzahl Source-Hosts berechnet werden. Das wird durch die Formel

$$R = \sum_{H=1}^{M} S_H \times N \tag{3.2}$$

wiedergegeben, wobei N die Anzahl an Rechnern ist, welche eine Verbindung aufbauen können müssen.

Die Anzahl Regeln würde sich also bei jedem neuen Host oder Dienst vervielfachen, wie Tabelle 3.1 zeigt. Auch müsste jede NAT-Firewall im gesamten Überwachungs-Netzwerk neu konfiguriert und eingestellt werden.

Anzahl Dienste	Anzahl entfernte Hosts	Anzahl NAT-Regeln
10	1	10
10	2	20
10	5	50
10	20	200
10	21	210
10	50	500
10	51	510

Tabelle 3.1: Anzahl NAT-Regeln bei gleich bleibender Anzahl zu überwachenden Dienste und steigender Anzahl von Hosts, welche diese Dienste überwachen

3.1.2 Umgehung der NAT-Problematik

Um die NAT-Regeln und den damit zugrunde liegenden Aufwand zu verringern, wäre eine Möglichkeit, dass alle Netzwerke in sich ein geschlossenes System bilden. Die Systeme in einem Netzwerk überwachen also alle Dienste und Komponenten ihres eigenen Netzwerkes. Um die Erreichbarkeit der einzelnen Netzwerke zu überwachen, werden

[7] Der Source-Host ist ein System, welches eine Anfrage startet. Der Destination-Host ist das System, an welches die Anfrage gerichtet wird

diese in der Theorie ebenfalls als einzelne Systeme angeschaut, welche sich gegenseitig überwachen (siehe Abbildung 5.2 im Anhang). Es werden also zwei verschiedene „Überwachungsnetzwerke“ definiert: Die einzelnen Komponenten innerhalb eines Netzwerkes überwachen sich gegenseitig; Die verschiedenen Netzwerke werden als einzelne Systeme angesehen und überwachen sich gegenseitig.

Die Überwachung von öffentlichen Diensten wie Web und EMail stellt dabei kein grosses Problem dar, denn diese sind wahrscheinlich von überall her erreichbar und somit auch überwachbar. Die interessantere Frage in diesem Kontext ist vielmehr, wie man trotz Firewalls prüfen kann, ob ein Netzwerk verfügbar/erreichbar ist oder nicht.

ICMP-Pakete

ICMP[Pos81][8] zeichnet sich durch sehr kleine und einfache Pakete aus, welche alle notwendigen Informationen zur Bestimmung von Latenzen sowie der Erreichbarkeit eines Systems beinhalten. Viele Überwachungs-Tools setzen aus diesem Grund auch ICMP für diese Zwecke ein. Der Nachteil ist, dass diese Pakete vielfach von Firewalls geblockt oder als unzustellbar zurückgeschickt werden[9]. Dies kann jedoch umgangen werden, indem pro Netzwerk eine Regel definiert wird, welche ICMP zulassen würde.

Eine ICMP-Prüfung kann nur gewährleisten, dass der Router (respektive der Gateway) des entfernten Netzwerkes augenscheinlich vorhanden ist und auf den ECHO-Request Antworten kann. Es können weder Daten noch Statusmeldungen etc. über dieses Protokoll ausgetauscht werden.

Überwachungs-Agenten

Eine andere Möglichkeit um Prüfungen zwischen verschiedenen Netzwerken zu realisieren sind spezielle Proxies, respektive Agenten. Ein Agent kann eine Nachricht empfangen, an ein System weiterleiten und gegebenenfalls dessen Antwort wieder zurücksenden.

Die Problematik bei einem einzelnen Agenten pro Netzwerk ist jedoch, dass wenn dieser nicht läuft, es den Anschein macht, dass das komplette Netzwerk nicht online ist. Für diesen Fall, also wenn ein Proxy keine Antwort sendet, könnte die ICMP-Lösung

[8] Mit ICMP oder auch PING wird umgangssprachlich ein ECHO-Request in einem IPv4 oder IPv6 Netzwerk betitelt.

[9] Die Ablehnung von ICMP-Paketen macht zum Beispiel Sinn, wenn man nicht will, dass andere auf eine schnelle Art und Weise herausfinden können, dass etwas da ist. Ein Portscan (auffinden eines offenen Ports bei einem System) dauert einiges länger und wird von den meisten Firewalls erkannt und somit ebenfalls geblockt

(siehe Kapitel 3.1.2) als Fallback-Lösung[10] eingesetzt werden. Dadurch weiss man anschliessend jedoch nur, ob wirklich das Netzwerk oder einfach nur der Agent nicht online ist.

Eine weitaus bessere Möglichkeit als der ICMP-Fallback ist es, pro Netzwerk mehrere Agenten zu definieren. Wenn von einem Agenten keine Antwort kommt, kann der zweite und dritte Agent noch geprüft werden. Wenn keiner der Agenten eine Antwort gibt, kann mit einer gewissen Wahrscheinlichkeit davon ausgegangen werden, dass entweder das Netzwerk die Internetverbindung verloren hat oder dass ein grösseres Problem in dem entfernten Netzwerk anliegt. Damit der administrative Aufwand bei der Firewall-Konfiguration nicht zu gross wird, müssen diese Agenten über eine Authentifizierung und gegebenenfalls sogar über eine Verschlüsselung verfügen. Dadurch müssen die NAT-Regeln nicht auf alle Source-Adressen gebunden werden, sondern es kann eine Verbindung von überall aus erlaubt werden.

SSL-VPN / OpenVPN

Während man bei traditionellen VPN-Lösungen fast ausschliesslich nur mit vorkonfigurierten Clients auf meistens nur einen VPN-Server zugreifen kann, bietet eine SSL-VPN Installation eine gewisse Freiheit. SSL-VPN findet heute[11] wieder grosse Beliebtheit, gerade für die Vernetzung von mobilen Endgeräten in Firmennetzwerke. Dabei wird bei SSL-VPN die komplette Authentifizierung direkt bei der Verbindung ausgehandelt, muss auf der Client-Seite also nicht voreingestellt werden. Dieser Vorteil ist zugleich auch ein Nachteil, denn die Verbindung kann von jedem Gerät getätigt werden, ob dies nun ein öffentlicher Computer in einem Internet-Kaffee oder ein Viren verseuchtes Smartphone eines Bekannten ist.

Für die Verbindung von mehreren Netzwerken zum Zwecke der Überwachung, spielen diese Nachteile jedoch keine Rolle. Pro Netzwerk, ob dieses nun durch NAT abgesichert ist oder nicht, werden verschiedene Systeme definiert, welche als SSL-VPN Server fungieren. Mehrere Systeme sollten es sein, damit bei einem Ausfall eines Servers ein Anderer den Dienst übernehmen kann. Sofern die eingesetzte Firewall SSL-VPN unterstützt, kann diese auch den Dienst als VPN-Server übernehmen. Denn fällt die Firewall aus, ist meistens so oder so das komplette Netzwerk nicht mehr funktionsfähig.

Jeder Server, welcher nun ein System aus einem anderen Netzwerk überwachen soll,

[10] Bei ICMP als Fallback-Lösung kommt wieder das Problem zum tragen, dass ECHO-Requests vielfach auf den Firewalls geblockt/abgewiesen werden

[11] In der Vergangenheit wurde SSL-VPN vielfach eingesetzt für die Authentifizierung beim Aufbau einer VPN-Verbindung, wurde jedoch durch IpSec immer mehr verdrängt

kann entweder eine neue Verbindung aufbauen oder eine schon bestehende Verbindung eines anderen Systems nutzen. Bei SSL-VPN besteht die Möglichkeit, einen einzelnen Rechner in ein Netzwerk einzubinden (Client-to-Site) oder aber zwei Netzwerke zu verbinden (Site-to-Site). Werden zwei Netzwerke miteinander verbunden, existiert pro Netwzerk je ein VPN-Gateway, auch Router genannt. Dieser Router hat die Aufgabe, alle Anfragen welche an das andere Netzwerk gerichtet sind, an eben dieses Netzwerk zu senden. Diese Verbindung zwischen den zwei VPN-Gateways wird auch VPN-Tunnel genannt, da alle Netzwerkpakete von einem Punkt zu einem anderen geschickt werden und dort anschliessend weiter verteilt werden. Nur die Definition „VPN“ bedeutet nicht, dass dieser Tunnel verschlüsselt und somit sicher ist, dies wird erst durch SSL (oder IpSec) gewährleistet.

OpenVPN[ope10b] bietet in diesem Bereich eine gute Lösung, welche durch die GPL-v2 Lizenzierung[12] auch in eigene Projekte integriert werden darf. Dadurch kann nicht nur der Client mit der SSL-VPN Technologie ausgestattet und ausgeliefert, sondern der SSL-VPN Serverbereich kann ebenfalls direkt mit dem Produkt ausgeliefert werden.

3.1.3 Fazit: Problemfall NAT

Die einfachste und komfortabelste Lösung stellt eindeutig die SSL-VPN Lösung von OpenVPN dar. Auf einer SSL-VPN fähigen Firewall müssen lediglich die Konfigurationen vorgenommen werden, damit die Hosts sich anmelden dürfen; bei den anderen müssen nur NAT Regeln zu zwei oder drei internen Servern freigeschalteten werden.

Die Verbindung der verschiedenen Netzwerke kann dadurch komplett autonom und automatisch erfolgen. Die einzigen Parameter, die in der Konfiguration vorhanden sein müssen, sind die Portangaben inklusive dem entfernten Gateway, auf welchen der/die OpenVPN Server laufen sowie die Benutzerangaben oder ein X.509-Zertifikat[13].

Zusätzlich wird auch noch ICMP eingesetzt, um die reine Verfügbarkeit der unterschiedlichen Netzwerke zu prüfen. Wird entdeckt, dass ein Netzwerk per ICMP nicht verfügbar ist (geblockt durch die Firewall), wird angedacht, einen SNMP-Trap basierenden Ping-Trapping-Dienst anstelle dessen zu starten. Mehr zu SNMP-Traps wird in Kapitel 3.6.1 beschrieben, unter anderem auch die Problematik welche sich dahinter verbirgt.

[12] Die GPL-v2 Lizenzierung ist eine Software-Lizenz, welche es erlaubt, den Source-Code komplett zu kopieren zu verändern und weiter zu verwenden, jedoch mit der Auflage, dass der Code immer frei verfügbar sei muss und nicht verkauft werden darf.

[13] X.509 ist ein Standard für eine Public-Key-Infrastruktur. Der Begiff X.509-Zertifikat bezieht sich in diesem Kontext auf den IETF-Standard von RFC-5280[CSF+08], also die Daten, welche in einem Zertifikat angegeben werden müssen und können.

Management-Netzwerk

Viele Server und Netzwerkkomponenten verfügen in der heutigen Zeit über Management-Interfaces. Mit diesen zusätzlichen Netzwerk-Schnittstellen wird, neben dem operativen, ein Management-Netzwerk aufgebaut, welches lediglich für die Administration und Überwachung da ist. Während grosse Netzwerke von Firmen und Hostern bereits so überwacht und administriert werden, wäre dieser Aufwand für kleine Firmen und vor allem bei kleinen Netzwerken schlicht zu gross. Auch die jeweils doppelte Anbindung von Systemen und Netzwerken in anderen Filialen, durch unterschiedliche Internetdienstleister, wäre wohl zu viel des Guten. Die Software sollte dennoch auf diesem Prinzip aufbauen und die Möglichkeit aufweisen, ein solches Management-Netzwerk zu simulieren. Durch OpenVPN können verschiedene Netzwerke einfach und schnell verbunden und konfiguriert werden, so also ein Management-Netzwerk simuliert werden. Zudem steht OpenVPN unter der GPLv2, kann also in der eigenen Software integriert und mit dieser ausgeliefert werden.

In der TechDemo, welche im Rahmen dieser Arbeit ausgearbeitet wird, wird nicht zwischen einem Operativem- und Management-Netzwerk unterschieden, da dies kein garvierender Punkt in der Machbarkeit einer solchen Software darstellt. In einer zukünftigen Version muss jedoch die Möglichkeit bestehen, ein Management-Netzwerk mittels OpenVPN auf zu bauen, sofern physisch kein solches vorhanden ist. Dadurch kann die Überwachungssoftware schon von Anfang an „physisch“ getrennt eingesetzt werden und es muss bei einer Expansion, respektive einer physischen Implementierung eines Management-Netzwerkes, nicht auch noch das komplette Überwachungsnetzwerk angepasst werden.

3.2 Problemfall Geräteerkennung

Grundsätzlich gibt es zwei Möglichkeiten Geräte in eine Überwachung ein zu schliessen. Entweder werden alle Geräte, welche überwacht werden sollen, manuell eingepflegt und konfiguriert oder diese werden automatisch aufgespürt, untersucht und anschliessend in der Konfiguraton eingepflegt.

3.2.1 Manuelle Konfiguration

Bei einer manuellen Konfiguration müssen alle Systeme von Hand konfiguriert werden. Dies kann mittels einer Konfigurationsdatei oder auch über eine grafische Oberfläche erfolgen. Dabei müssen alle Systeme, seien dies nun Überwachungsserver oder normale

Komponenten, aus allen Netzwerken definiert und eingetragen werden. Dies gilt auch für die verschiedenen Dienste (SNMP, HTTP, SMTP, etc.), welche auf den verschiedenen Servern und Systemen vorhanden sind. Zusätzlich müssen auch die verschiedenen Netzwerke sowie die Verbindung zwischen diesen definiert werden.

Die so erstellte Konfiguration muss anschliessend auf die verschiedenen Überwachungsserver verteilt werden. Bei einer rein manuellen Konfiguration kann dies über ein manuelles Kopierverfahren gemacht werden. Dadurch besteht aber die Gefahr der Inkonsistenz, also dass jeder Überwachungsserver eine unterschiedliche Konfiguration hat. Aus diesem Grund sollte die Verteilung der Konfiguration automatisch erfolgen. Hierfür eignen sich Methoden und Techniken, wie sie unter 3.2.2 beschrieben werden.

3.2.2 Auto-Discovery

Ein gutes Beispiel bezüglich AutoDiscovery bietet NeDi (siehe 2.1.7). Alternativ zu CDP und LLDP, welches die Geräte unterstützen müssen, kann auch direkt ARP oder ICMP als Scan-Methode eingesetzt werden.

Der Vorteil von ARP gegenüber ICMP liegt darin, dass ARP grundsätzlich immer zur Lokalisierung eines Hosts gemacht wird. Soll Beispielsweise ein ICMP Paket gesendet werden, muss das Betriebssystem zuerst die Hardware-Adresse des Ziels ausfindig machen. Es wird also zuerst ein ARP-Paket auf das lokale Netzwerk geschickt (Broadcast), welches die Ziel-IP-Adresse beinhaltet. Ist der Host verfügbar, antwortet dieser mit seiner MAC-Adresse. Somit ist bekannt, dass der Host verfügbar ist und auch welche Hardware-Adresse er hat. Das ICMP-Paket kann anschliessend an die empfangene MAC-Adresse gesendet und die Antwort ausgewertet werden.

Der grosse Nachteil von ARP ist jedoch, dass nur Systeme im eigenen Netzwerk aufgefunden werden können. Sollen auch Systeme in anderen Netzwerken automatisch gefunden werden, muss an Stelle von ARP dennoch ICMP verwendet werden.

Tests zufolge sind nicht alle Betriebssysteme darauf ausgelegt, Hunderte oder gar Millionen von ARP- und ICMP-Anfragen gleichzeitig auf das Netzwerk zu legen, die Antworten ab zu warten und aus zu werten. Eine performante und gut getestete Implementierung des ARP-Discovery, Betriebssystem unabhängig, wird in NMAP[14] eingesetzt. Bei der Implementierung der Auto-Discovery Funktion sollte wenn möglich auf die bei NMAP verwendeten Algorithmen und Bibliotheken zurückgegriffen werden.

Die automatische Erkennung von Systemen sollte grundsätzlich immer nur pro Netzwerk gemacht werden. Dabei kann jedoch zwischen verschiedenen internen und den ex-

[14]NMAP ist ein freies Tool zur Netzwerk Analyse und für Sicherheits-Audits. Mehr auf http://www.nmap.org/ im Bereich „Host-Discovery“

ternen Netzwerken unterschieden werden. Interne Netzwerke können, sofern diese über eine direkte Verbindung/Route verfügen, auch gemeinsam untersucht und überwacht werden. Wenn verschiedene Netzwerke jedoch mittels NAT getrennt sind, müssen diese getrennt voneinander untersucht werden. Dabei muss in jedem der verschiedenen Netzwerke mindestens ein Überwachungsserver mit einer entsprechend konfigurierten Verbindung vorhanden sein (siehe 3.1.3).

Nachdem alle verfügbaren Systeme gefunden sind, werden diese der Reihe nach untersucht: Durch den Aufbau von Verbindungen zu verschiedenen Diensten, kann gesagt werden, welche von diesen auf einem System laufen und welche nicht. Ist das aktuelle System ein Überwachungsserver, wird dieses zusätzlich markiert. Bei den Überwachungservern wird zusätzlich geprüft, ob diese eine Verbindung in ein anderes Netzwerk besitzen. Wenn dies der Fall ist, teilt der Überwachungsserver diese Daten des entfernten Netzwerkes allen Systemen mit. Anschliessend bekommt das System den Auftrag, das entfernte Netzwerk zu untersuchen. Es prüft dabei zuerst ob das entfernte Netzwerk untersucht werden darf und nicht bereits untersucht worden ist. Darf das Netzwerk untersucht werden und dies noch nicht gemacht worden ist, startet das Auto-Discovery im entfernten Netzwerk. Ein schematischer Ablauf dieses Vorgangs ist in Abbildung 3.1 zu sehen.

3.2.3 Fazit: Problemfall Geräteerkennung

In grösseren Netzwerken oder Verbünden verschiedener Netzwerke ist eine reine manuelle Konfiguration sehr aufwändig und daher praktisch unmöglich. Eine reine automatische Erkennung von Geräten bietet jedoch eine zu geringe Möglichkeit zur individuellen Konfiguration der einzelnen Systeme.

Eine Mischform aus einer manuellen und einer automatischen Konfiguration ist die einfachste und komfortabelste Lösung zur Einrichtung eines Überwachungsnetzwerkes. Dabei können einzelne Systeme jederzeit manuell in ein Netzwerk eingebunden, umkonfiguriert und ausgeschlossen werden, jedoch kann in einem Netzwerk auch automatisch nach neuen Komponenten gesucht werden.

Das zu implementierende System muss also über beide Möglichkeiten verfügen. Zum einen können die Überwachungsserver sowie verschiedene Komponenten in einem Netzwerk manuell konfiguriert werden, jedoch besteht auch die Möglichkeit dieses automatisch zu untersuchen. Dabei muss auf den Überwachungsservern manuell definiert werden, welche Netzwerke miteinander verbunden sind und welche von diesen automatisch untersucht werden sollen, respektive welche nicht[15]. Diese Konfiguration muss nicht auf

[15]Der Ausschluss eines Netzwerkes macht zum Beispiel bei Root-Servern Sinn, welche bei einem Hoster

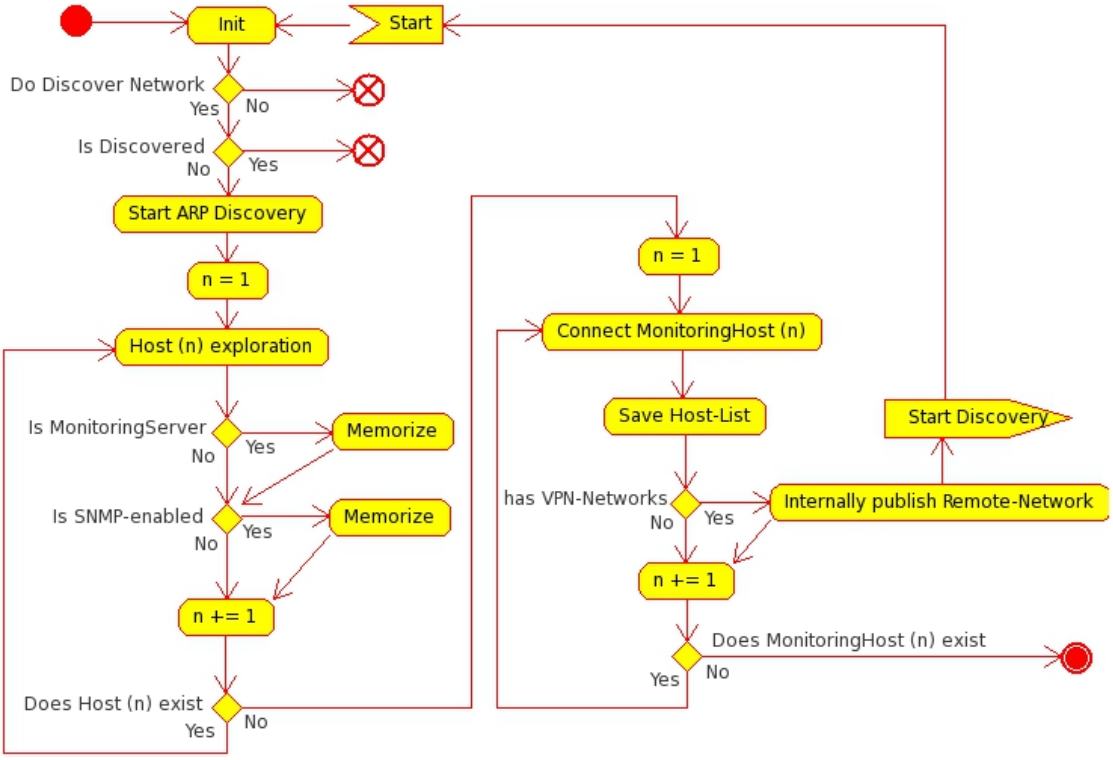

Abbildung 3.1: Automatische Suche nach allen Hosts im internen wie auch in entfernten Netzwerken. Dieses wird nur neu ausgelesen, sofern dies nicht schon gemacht worden ist. Dadurch wird verhindert, dass ein Netzwerk von mehreren Überwachungsservern nacheinander oder gleichzeitig ausgelesen wird.

jedem Überwachungsserver in jedem Netzwerk gemacht werden. Es reicht dabei aus, wenn diese Konfiguration auf einem einzelnen Server gemacht wird, denn die Konfiguration wird automatisch auf alle bekannten Überwachungsserver übertragen (siehe 3.3.2).

3.3 Problemfall Konfiguration

Die verschiedenen Überwachungsserver können, wie zum Beispiel unter 3.2.3 definiert, unterschiedlich und individuell konfiguriert werden. Das gesamte Überwachungsnetzwerk kann also auf jedem X-Beliebigen Überwachungsserver verändert werden. Diese Konfiguration muss anschliessend nicht nur auf alle anderen Systeme übertragen sondern auch abgeglichen werden.

direkt im Internet sind, dort jedoch nicht das komplette Netzwerk, sondern nur eben der eine (oder mehrere) Server überwacht werden soll.

3.3.1 Möglichkeiten zur Konfigurationsspeicherung

Grundsätzlich kann zwischen einer globalen und einer individuellen Konfiguration unterschieden werden.

Global zusammengefasste Konfiguration

Eine Möglichkeit besteht darin, dass jeder Überwachungsserver jeweils die komplette Konfiguration speichert.

Der Vorteil bei dieser Art der Speicherung liegt in der einfachen Handhabung, denn es gibt nur eine einzige Konfiguration. Diese kann auf einem beliebigen System angepasst und anschliessend 1:1 an alle anderen Server wieder ausgeliefert werden.

Nachteilig an dieser Variante ist jedoch, dass bei einem grossen Überwachungsnetzwerk die Konfiguration relativ gross werden kann. Je nach Anzahl Komponenten und Diensten wird diese Art der Konfiguration relativ schnell komplex und wahrscheinlich fast nicht mehr überschaubar (ohne entsprechende Konfigurations-Hilfen/-Programme).

Individuelle Konfiguration

Die zweite Möglichkeit beinhaltet eine individuelle Konfiguration auf jedem einzelnen System. Jeder Überwachungsserver besitzt somit nur seine jeweilige Konfiguration und weiss dadurch auch nur, was er zu überwachen hat. Die restlichen Systeme und Netzwerke sind ihm nicht bekannt.

Der Vorteil dieser Variante liegt eindeutig in der Konfigurationsgrösse, welche relativ klein und pro System sehr übersichtlich ist.

Der grosse Nachteil liegt im Management der verschiedenen Konfigurationen. Für einen globalen Überblick über die komplette Konfiguration, oder auch nur darüber, welche Server den einen Dienst auf einem Server überwacht, muss die komplette Konfiguration aus dem ganzen Überwachungsnetzwerk zusammengesucht, kombiniert und ausgewertet werden. Für diesen Vorgang müssen nicht nur alle Überwachungsserver im eigenen Netzwerk bekannt sein und abgefragt werden sondern auch alle in allen anderen Netzwerken.

3.3.2 Fazit: Problemfall Konfiguration

Das zu entwickelnde Überwachungssystem soll sich automatisch, autonom und vor allem hochverfügbar überwachen. Eine Voraussetzung ist somit, eine schnelle und vor allem unkomplizierte Auswahl von verschiedenen Überwachungspartnern. Dies kann nur mit einer global zusammengefassten Konfiguration erreicht werden.

Die Performance der einzelnen Systeme ist auch eine Voraussetzung, gerade im Bereich der Hochverfügbarkeit. Dieser Aspekt kann jedoch nur durch eine individuelle lokale Konfiguration gewährleistet werden.

Durch diese Voraussetzungen kommt nur eine Kombination der beiden Varianten in Frage. Zusätzlich wird die Konfiguration noch in verschiedene Bereiche aufgeteilt, welche separiert gespeichert und abgefragt werden können.

Konfiguration der Netzwerke

Die verschiedenen Netzwerke sowie die Verbindungen zwischen diesen werden in einer globalen, netzwerkübergreifenden Konfiguration gespeichert. Dadurch wissen immer alle Überwachungssysteme, welche anderen Netzwerke überwacht werden und wie auf diese zugegriffen wird. Diese Konfiguration beinhaltet nur die Netzwerke und deren Daten zur Fragmentierung, Angaben ob das Netzwerk automatisch untersucht werden darf (siehe 3.2.3), Verbindungsdaten für die Verbindungen (siehe 3.1.3) sowie eventuelle weitere Daten, welche auf den Netzwerken definiert werden müssen. Dies können zum Beispiel Dienste sein, welche von ausserhalb erreichbar sein müssen, etc.

Diese globale Konfiguration ist nicht als eine zentrale Datenbank zu sehen sondern als eine Konfiguration, welche auf jedem System gleich ist. Bei einer Anpassung wird somit jedes System mit der neuen Konfiguration versehen und nicht nur das betroffene System.

Konfiguration der Hosts und Systeme

Die verfügbaren Hosts sowie deren Dienste, die überwacht werden sollen, sind jeweils nur Netzwerkintern als globale Konfiguration vorhanden. Diese Konfiguration wird jeweils bei einer automatischen Geräteerkennung und auch einer manuellen Anpassung neu geschrieben und verteilt. Auch diese globale Konfiguration ist nicht eine zentrale Datenbank, sondern grundsätzlich einfach auf jedem System gleich.

Interne Dienste, wie beispielsweise die Überwachung des Arbeitsspeichers oder der Systemlast, werden gleich behandelt wie externe Dienste. Alle internen Dienste sollten ebenfalls über einen SNMP-Dienst von extern abrufbar sein.

Durch diese Konfiguration weiss jedes System, was auf welchem anderen System laufen muss, was für Probleme auftreten können und was bei einem Fehler passieren soll.

Konfiguration der Überwachungspartner

Die Auswahl der Überwachungspartner muss automatisch und autonom erfolgen (siehe 3.4). Die Konfiguration, welches System welche Hosts und Dienste überwacht, wird je-

weils auf den Systemen und nicht global gespeichert. Somit kann schnell eruiert werden, wann was wo überwacht werden soll, ohne zuerst andere Systeme an zu fragen, ob dies noch der Fall ist oder ob sich etwas geändert hat.

Diese Konfiguration wird automatisch erstellt, gewartet und gelöscht. Sie kann nicht manuell bearbeitet werden.

3.4 Problemfall Host-Fragmentierung

Sind alle Systeme gefunden und untersucht, muss in einem nächsten Schritt festgelegt werden, welche Überwachungsserver welche anderen Systeme überwachen sollen.

Die Grundfrage „Wer überwacht wen?“ beinhaltet folgende weiteren Fragestellungen, welche bei den verschiedenen Methoden zur Hostfragmentierung geklärt werden müssen:

- Werden die Dienste nur Netzwerk intern überwacht oder auch von externen Systemen?
- Wie viele Server überwachen einen einzelnen Dienst auf einem System?
- Können verschiedene Überwachungen zu unterschiedlichen Zeitpunkten gemacht werden?
- Darf ein Server denjenigen überwachen, welcher einen seiner Dienste überwacht?
- Überwacht ein Server nur einen einzelnen Server oder überwacht er verschiedene Systeme?
- Überwacht ein Server alle Dienste eines anderen oder nur einzelne?

In Kapitel 3.1.3 wurde die Verbindung zwischen unterschiedlichen Netzwerken behandelt. Dadurch ist die erste Frage schon abgehandelt. In jedem Netzwerk steht mindestens ein Überwachungsserver, welcher eine Verbindung in die anderen Netzwerke hat. Diese Server sind dafür zuständig, die entfernten Netzwerke zu überwachen. Das beinhaltet nicht nur die Erreichbarkeit, sondern auch die Überwachung aller Dienste, welche als „von extern zu überwachen“ konfiguriert worden sind (siehe Kapitel 3.3.2).

Die Frage nach der Anzahl Server, welche einen einzelnen Dienst auf einem System überwachen, kann einfach beantwortet werden: Durch die Definition einer hochverfügbaren Überwachung, sollte ein Dienst, wenn möglich, immer von mehreren anderen Systemen überwacht werden und nicht nur von einem einzelnen. Fällt ein System aus, wird die Überwachung somit immer noch von mindestens einem anderen Gerät durchgeführt.

Die restlichen Fragen gehen in den Bereich der Performance und der Lastverteilung.

3.4.1 Lastverteilung

Die Systemlast sollte, gerade da es sich um eine hochverfügbare Überwachung handelt und die Server nicht zusätzlich belastet werden sollen, möglichst gering sein und somit eine hohe Performance aufweisen. Die zu überwachenden Dienste müssen also möglichst gleichverteilt auf alle internen Überwachungsserver verteilt werden. Durch diese Gleichverteilung kann davon ausgegangen werden, dass jeder Überwachungsserver eine ähnliche, zusätzliche Grundlast aufweist. Mit der Formel

$$L = X \times N \tag{3.3}$$

kann berechnet werden, was für eine zusätzliche Last L ein Überwachungsserver verkraften muss, wenn je nur ein einzelner Dienst von allen Servern überwacht wird. Dabei wird für X die Last des jeweiligen Dienstes und für N die Anzahl überwachter Server genommen.

	HTTP (50)	SMTP(10)	ICMP (5)	POP3 (30)
Server A	x	x	x	x
Server B	x		x	
Server C			x	
Server D	x	x	x	x
Server E	x		x	
Total	4	2	5	2

Tabelle 3.2: Beispiel: Verschiedene Server mit unterschiedlichen Diensten, die überwacht werden sollen. In Klammern ein Dummy-Wert zur Berechnung, welcher die zusätzliche „Last“ einer Überwachung angibt. Eine HTTP-Dienstüberwachung mit Result-Auswertung braucht mehr Resourcen als eine einfacher ICMP-EchoRequest.

	HTTP (50)	SMTP(10)	ICMP (5)	POP3 (30)	Zusätzliche Last
Server A	4				200
Server B		2			20
Server C			5		25
Server D				2	60
Server E					0

Tabelle 3.3: Beispiel: Jeder Server übernimmt die Überwachung von einem einzelnen Dienst. In Klammern ein Dummy-Wert zur Berechnung, welcher die zusätzliche „Last“ einer Überwachung angibt. Jeder Server hat somit eine stark unterschiedliche Zusatzlast, mit Ausnahme von Server E, welcher nicht weiter belastet wird. Problematisch sind nicht nur die unterschidlichen Lasten, sondern auch dass die Server ihre eigenen Dienste überwachen müssen.

Damit die zusätzliche Grundlast möglichst gleichverteilt auf allen Systemen ist, sollte

eine möglichst gemischte Dienstüberwachung gewählt werden. Die verschiedenen Dienstüberwachungen weisen wahrscheinlich unterschiedliche Lasten in der Verarbeitung auf, denn bei den einen Diensten muss lediglich die Erreichbarkeit geklärt werden, bei anderen müssen eventuell komplexe Stringmanipulationen oder verschiedene Sende- und Empfangs-Aktionen durchgeführt werden. Durch die Definition, dass ein Überwachungsserver unterschiedliche Dienste überwacht, kann also sichergestellt werden, dass eine Lastverteilung stattfindet. Mit der Formel

$$L = \sum_{i=0}^{N} X_i \tag{3.4}$$

kann diese Zusatzlast L, welche ein Server verkraften muss, berechnet werden. Dabei ist i ein Zähler von 0 bis zur Anzahl Dienste N und X_i die jeweilige Last des aktuellen Dienstes.

Für eine genaue Berechnung und optimale Lastverteilung ist es dabei notwendig, dass die Summe der Last der Dienste gleichmässig auf alle Server verteilt werden kann. Es müsste also „$AnzahlDienste \mod AnzahlServer = 0$“ gegeben sein, um eine optimale Verteilung zu gewährleisten. Diese Definition sagt aus, dass jedem Server gleich viele Dienste zugewiesen werden können, die Berechnung $\frac{AnzahlDienste}{AnzahlServer}$ also keinen Rest ergibt - Wobei bei dieser Definition die Last der Dienste nicht beachtet wird. Ein Vergleich zwischen den zwei Varianten ist, basierend auf den Daten aus Tabelle 3.2, in den Tabellen 3.3 und 3.4 zu sehen.

	HTTP (50)	SMTP(10)	ICMP (5)	POP3 (30)	Zusätzliche Last
Server A	1		1		55
Server B	1	1	1		65
Server C	1		1	1	85
Server D	1		1		55
Server E		1	1	1	45

Tabelle 3.4: Beispiel: Die 13 Dienste werden auf die verschiedenen Server aufgeteilt. Somit werden die Lasten nahezu gleich auf alle Server verteilt.

Eine andere Möglichkeit besteht darin, dass ein Überwachungsserver jeweils alle Dienste von einem oder von mehreren anderen Systemen überwacht. Die dabei anfallende Last kann mit der Formel

$$L = \sum_{k=0}^{M} \left(\sum_{i=0}^{N} X_{k_i} \right) \tag{3.5}$$

berechnet werden, wobei M die Anzahl der Server, N die Anzahl der Dienste auf dem

Server M_k und X_{k_i} die Last des Dienstes X_i auf dem aktuellen Server ist. Tabelle 3.5 zeigt die Systemlasten, welche bei einer solchen Überwachung anfallen würden.

	HTTP (50)	SMTP(10)	ICMP (5)	POP3 (30)	Total Last
Server A	1	1	1	1	95
Server B	1		1		55
Server C			1		5
Server D	1	1	1	1	95
Server E	1		1		55

Tabelle 3.5: Beispiel: Alle Dienste auf einem System werden von einem Server überwacht. Die Tabelle zeigt die Lasten, welche beim Überwachungs-Server anfallen würden.

Da ein System meistens mehrere andere Rechner überwacht, muss bei dieser Variante darauf geachtet werden, dass die Auswahl der unterschiedlichen Systeme zu einer gleichverteilten Last führt. Die zusätzliche Last sollte also bei jedem Überwachungsserver um den Mittelwert aller Lasten liegen. Um dies zu erreichen, kann ein einfacher Algorithmus (siehe Codeblock 3.1) angewendet werden.

```
1:  if numMonitor ≥ 1 and numMonitor ≤ length(hosts)−1 then
2:    result {}
3:  end
4:
5:  if count(hosts) = 2 then
6:    result { h1:[h2], h2:[h1] }
7:  end
8:
9:  back = {}
10: intensiveLoad = floor(numMonitor / 2)
11: extensiveLoad = intensiveLoad
12:
13: if numMonitor modulo 2 ≠ 0 then
14:   extensiveLoad = intensiveLoad + 1
15: end
16:
17: hosts = sort_ascending_by_load(hosts)
18: monitoredHosts = []
19:
20: for host in hosts do
21:   if length(monitoredHosts) = length(hosts) then
22:     monitoredHosts = []
23:   end
24:
25:   x = 0
26:   while length(back[host]) ≠ intensiveLoad do
27:     if length(monitoredHosts) = length(hosts) break
28:     if hosts[x] ≠ host and hosts[x] not in monitoredHosts then
29:       add hosts[x] to back[host]
30:       add hosts[x] to monitoredHosts
31:     end
32:     x += 1
33:   end
34:
35:   x = length(hosts) − 1
36:   while length(back[host]) ≠ intensiveLoad + extensiveLoad do
37:     if length(monitoredHosts) = length(hosts) break
38:     if hosts[x] ≠ host and hosts[x] not in monitoredHosts then
39:       add hosts[x] to back[host]
40:       add hosts[x] to monitoredHosts
41:     end
42:     x −= 1
43:   end
44: end
45:
46: result back
```

Codeblock 3.1: Mit diesem Algorithmus werden alle zu überwachenden Dienste und Server gleichverteilt auf alle Überwachungsserver verteilt. Die Server liegen dabei als Liste $hosts \Rightarrow [h_1, h_2, ..., h_k]$ vor. Die Hosts werden der Last nach sortiert und anschliessend immer die beiden äussersten Systeme (also der Last-Intensivste und -Extensivste) einem Host zugewiesen.

3.4.2 Anzahl Überwachungspartner

Durch die Anzahl der Überwachungspartner wird nicht nur die Grundlast, welche auf den Servern anfällt, definiert, sondern auch die zusätzliche Last im Netzwerk, sowie der gesamte Vernetzungsplan. Durch die definierte Autonomität der Überwachungsstrategie, sollte also nicht eine fixe Anzahl an Verbindungen vorgegeben werden, sondern diese sollte automatisch berechnet werden.

Eine Möglichkeit bestünde darin, dass einfach jeder Rechner jeden anderen überwachen würde. Dies wäre jedoch nicht nur performancetechnisch Unsinn, sondern auch administrativ. Fällt beispielsweise in einem Netzwerk mit 100 Rechnern ein System aus, würden 99 andere Systeme das bemerken und müssten untereinander ausmachen, wer denn nun den Alert auslöst und versendet.

Es muss also ein Algorithmus entworfen/gefunden werden, welcher die ideale Anzahl an Verbindungen zwischen verschiedenen Knotenpunkten herstellt. Die Distanz zwischen zwei Punkten, respektive zwischen zwei Systemen kann dabei vernachlässigt werden, denn die Latenzen in einem internen Netzwerk müssen sowieso als sehr klein angenommen werden[16].

In der Informatik schon lange eingesetzte und bewährte Techniken bei Kommunikationsintensiven Abläufen sind die Hypercube-Algorithmen[17]. Die nachfolgenden Formeln sowie die Definitionen der Hamming-Distanz sind in vielen Fachliteraturen und im Internet verfügbar, zum Beispiel im Dokument von Hans Walser: Der n-dimensionale Hyperwürfel [Wal19]. Dieses Dokument diente als Grundlage für den Aufbau des grundlegenden Verständnisses der Hyperwürfel.

Ein Hypercube, auch Hyperwürfel genannt, stellt grundsätzlich ein Netzwerk aus Knoten und Verbindungen dar (siehe Abbildung 5.3 im Anhang). Der Einsatz der Hypercubes im Bereich des Hypercomputing oder auch Clustering rührt daher, dass grundsätzlich von jedem Knoten des Hypercubes eine Verbindung zu jedem anderen Knoten besteht. Die Anzahl an Verbindungen bleibt dabei jedoch überschaubar, da nicht einfach jeder Knoten mit jedem anderen verbunden wird. Es bestehen aber auch nicht zu wenige Verbindungen, so dass das Fehlen eines Systems nicht zu einem Totalausfall führt.

Bei einem Hypercube werden alle Systeme je einem Knoten zugewiesen. Die Distanz zwischen zwei Systemen ist im Optimalfall 1, im schlimmsten Fall gleich der Dimension n

[16] Sind die Latenzen in einem internen Netzwerk, also z.B. einem Gebäude, schlecht, stimmt entweder etwas an der Verkabelung nicht, das Netzwerk ist überlastet oder es werden alte Komponenten eingesetzt, welche mit der Last nicht zurecht kommen.

[17] Danke an dieser Stelle für den Hinweis auf die Hypercubes an Herr Dr. Frank Moehle.

des Hypercubes[18]. Diese Distanz wird auch *Hamming-Distanz*[19] genannt, entsprechend werden im Hypercube auch die verschiedenen Ecken nummeriert und verbunden (siehe Abbildung 5.4 im Anhang).

Zur Bestimmung der Dimension des Hypercubes muss die Anzahl der Systeme bekannt sein. Die Anzahl Ecken in einem Hypercube wird definiert durch 2^n, wobei n die Dimension ist. Es muss also die nächst höhere Zweierpotenz gefunden werden, um die Dimension des Hypercubes zu bestimmen (siehe Codeblock 3.2). Basierend auf der Dimension können anschliessend alle weiteren Daten berechnet werden.

```
1:  // bin: 0000 0001
2:  power = 1
3:  if (numHosts > 1) {
4:    while (power < numHosts) {
5:      // Shift left as long as power is less than numHosts
6:      // bin: 0000 0010, 0000 0100, 0000 1000, ...
7:      power <<= 1
8:    }
9:  }
10: return power
```

Codeblock 3.2: Algorithmus zum Auffinden der nächst höheren Zweierpotenz zur Dimensionsbestimmung eines Hypercubes anhand der Anzahl Systeme. Die Schlaufe „schiebt" dabei die 1 in der binären Darstellung so lange nach links bis die Potenz grösser oder gleich der Anzahl Systeme ist.

Die Anzahl Überwachungspartner bei einem Hypercube ist immer gleich der Dimension. Somit kann durch den Algorithmus in Codeblock 3.1 bestimmt werden, welche Systeme welche anderen überwachen. Dies jedoch nur unter der Voraussetzung, dass jedes System das überwacht wird, ebenfalls ein Überwachungsserver ist. Das kann aber nicht als Voraussetzung gegeben werden, denn in einem Netzwerk werden oft Geräte eingesetzt, welche sich zwar überwachen lassen, selber jedoch nicht in der Lage sind, andere Systeme zu überwachen - geschweige denn es zulassen, Software nach zu installieren, welche diese Funktionalität übernehmen würde.

Eine Möglichkeit, dieses Problem zu umgehen wäre, eine spezielle Anordnung der Nodes im Hypercube. Dies würde eine relative Verkomplizierung des Algorithmus nach sich ziehen, denn bei jeder Platzierung müssten alle Nachbarknoten angeschaut werden und auf die Überwachungsfähigkeit untersucht werden.

[18] Die Dimension beim Hypercube ergibt sich aus der Anzahl von Systemen, die überwacht werden müssen siehe Abbildung 5.3 im Anhang

[19] Die *Hamming-Distanz* findet ihren Ursprung bei Richard Wesley Hamming, welcher durch den Hamming-Code eine Selbst-Korrigierende Übermittlungsmethode entwickelte.

Eine weitaus einfachere Möglichkeit ist, den Hypercube nur mit den Überwachungsservern als Knoten zu erstellen. Die restlichen Systeme werden anschliessend einfach harmonisch auf alle Überwachungsserver aufgeteilt.

Durch folgende Vorgehensweise kann schnell und einfach eine Überwachung basierend auf einem Hypercube aufgebaut werden:

1. Dimension des Hypercubes anhand aller Überwachungsserver feststellen
2. Hypercube mit allen Überwachungsservern erstellen
3. Überwachungspartner festlegen, die Anzahl ist gleich der Dimension
4. Die restlichen Systeme auf alle Überwachungsserver aufteilen

3.4.3 Leere Knoten im Hypercube

Die Vernetzung aller Überwachungsserver mittels einem Hypercube ist eine schnelle und nicht sehr rechenintensive Angelegenheit. Ein Hypercube, welcher basierend auf einer Anzahl Hosts aufgebaut wird, welche keine Zweierpotenz ist, weist aber eine grosse Unschönheit auf: leere Knoten. Gerade wenn ein Hypercube mit vielen Systemen erstellt wird, nimmt die Wahrscheinlichkeit zu, dass leere Knoten in einem Hypercube vorhanden sind.

Im schlimmsten Fall kann ein Node durch diese leeren Knoten so abgeschirmt werden, dass dieser nur Verbindungen zu leeren Knoten aufweist. Eine Illustrierung dieses Problems wird in Abbildung 3.2 gezeigt. Ein Hypercube, der basierend auf 9 Systemen aufgebaut wird, muss in der Dimension $d = 4$ erstellt werden. Das bedeutet, dass es total $2^4 - 9 = 16 - 9 = 7$ leere Knoten gibt und jeder Knoten 4 Verbindungen aufweist. Im extremsten Fall können so bis zu vier Nodes komplett abgeschirmt sein. Bei einer Verwendung des Hypercubes zur Gewährleistung von Verbindungen von jedem Node im Netzwerk zu jedem anderen, kann dies zu schwerwiegenden Problemen führen.

Beim Aufbauen eines Hypercubes muss also immer darauf geachtet werden, dass die Nodes nicht willkürlich im Hypercube verteilt werden, sondern dass diese der Reihe nach aufgefüllt werden. Dadurch kann gewährleistet werden, dass jeder Knoten mindestens eine Verbindung aufweist.

3.5 Problemfall ALERT-Übermittlung

Wird von einem System ein Fehler entdeckt, muss dies je nach Konfiguration einer entsprechenden Person oder Stelle gemeldet werden. Die Problematik liegt nun darin, dass

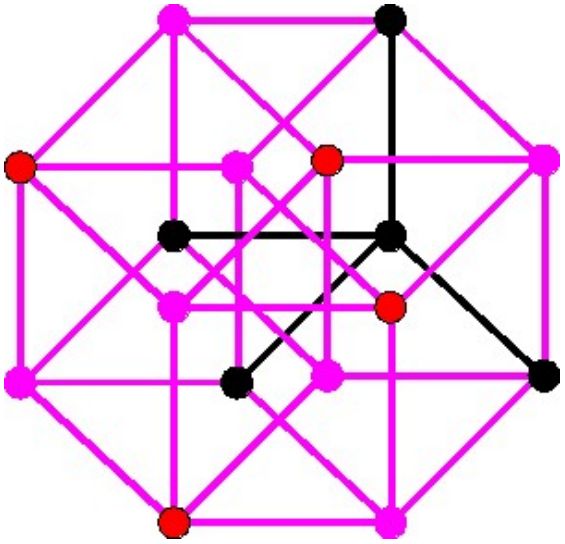

Abbildung 3.2: Ein Hypercube der vierten Dimension mit 9 Hosts (schwarz, rot) weist 7 leere Knoten auf (magenta). Bei einer unglücklichen Zuweisung der Nodes können so bis zu vier Systeme ohne Verbindung sein (rot).

ein System, respektive Dienst, von mehreren anderen Systemen zu unterschiedlichen Zeitpunkten überwacht wird. Eine Meldung sollte aber wenn möglich nur einmal versendet werden und nicht von jedem Überwachungssystem aus, welches den Fehler bemerkt.

3.5.1 Zentrale Meldestelle

Eine Möglichkeit diese mehrfache Alert-Auslieferung in den Griff zu bekommen, wäre eine zentrale Meldestelle. Jedes Problem, welches entdeckt wird, wird umgehend dieser zentralen Stelle gemeldet. Eine Meldung muss dabei mindestens die Zeit der Überwachung, den Meldungs-Typen sowie allfällige Daten beinhalten. In regelmässigen Abständen werden alle gespeicherten und nicht versendeten Meldungen ausgelesen, gruppiert und verschickt.

Die Gruppierung der Meldungen nach dem Zeitrahmen und Meldungs-Typ hat den Hintergrund, dass die verschiedenen Überwachungssysteme die Prüfungen zwar nicht zum gleichen Zeitpunkt dafür aber im gleichen Intervall[20] ausführen. Werden nun alle Nachrichten innerhalb eines solchen Intervalls ausgelesen (unter Beachtung des Systems und Dienstes), kann theoretisch ausgeschlossen werden, dass der gleiche Vorfall mehrfach gemeldet wird.

[20] Ein System soll zum Beispiel alle 5 Sekunden mit einem ECHO-Request auf Anwesenheit geprüft werden. Dieses Interval kann prinzipiell gewährleistet werden, nicht jedoch dass jedes System die gleiche Prüfung in der gleichen (Nano/Mili)Sekunde ausführt.

Der Vorteil einer solchen zentralen Instanz wäre klar die einfache Bearbeitung aller Fehler. Der Nachteil liegt jedoch in der Definition und der Auslegung des gesamten Überwachungs-Systems. Da das komplette System „hochverfügbar“ sein muss, wäre eine einzelne wenn auch nur Netzwerk interne Alert-Instanz ein Widerspruch sowie eine Fehlerquelle. Funktioniert diese zentrale Instanz nicht oder ist diese nicht erreichbar, können aus dem gesamten Netzwerk keine Fehler gemeldet werden.

3.5.2 Hochverfügbare Alert-Übermittlung

Um dem Paradigma der Hochverfügbarkeit gerecht zu werden, sollte prinzipiell jeder Überwachungsserver in der Lage sein, einen Fehler eigenständig zu melden. In diesem Szenario gibt es zur Vermeidung von mehrfachen Meldungen grundsätzlich zwei Möglichkeiten[21]:

POLL: Wird ein Fehler entdeckt, müssen alle anderen Systeme, welche das gleiche System und den gleichen Dienst überwachen, angefragt werden, ob dieser Fehler innerhalb der letzten X Zeiteinheiten schon gemeldet worden ist. Ist dies nicht der Fall, wird eine Meldung gesendet und der Status gespeichert.

PUSH: Wird ein Fehler entdeckt, müssen alle anderen Systeme benachrichtigt werden, dass dieser Fehler zum Zeitpunkt X gemeldet worden ist. Entdeckt ein anderes System den gleichen Fehler, kann es durch diese interne Liste prüfen, ob der gefundene Fehler innerhalb der letzten X Zeiteinheiten schon gemeldet worden ist oder nicht.

Bei beiden möglichen Szenarien besteht die Gefahr, dass das Netzwerk unnötigerweise belastet respektive überflutet wird. Jedes System setzt nach der offiziellen Überwachung noch weitere Anfragen an alle anderen Systeme ab. Die Anzahl der Anfragen[22] kann sich dadurch auf ein Maximum von $n * (n - 1)$ kumulieren.

Die Anfragen und Antworten werden wahrscheinlich relativ kleine Datenpakete sein. Die heutigen internen Netzwerke sind rasend schnell, jedoch sollen mit der angestrebten Lösung auch Systeme mit einer schwachen Leitung[23] und auch stark frequentierte und ausgelastete Netzwerke angesprochen und überwacht werden können. Bei der zu implementierenden Lösung muss dieser Punkt unbedingt beachtet werden.

[21] Diese zwei Möglichkeiten beschreiben die Basisfunktionalität - Kombinationen und Verfeinerungen, etc. sind möglich, werden aber nicht als weitere Möglichkeiten gelistet.

[22] Wenn n Systeme gleichzeitig an alle $n - 1$ anderen Systeme eine Anfrage stellen, sind gleichzeitig $n * (n - 1)$ Requests auf dem Netzwerk

[23] Zum Beispiel ein einzelnes externes System, welches lediglich über eine GSM-Verbindung verfügt

3.5.3 Fazit: Problemfall ALERT-Übermittlung

Das zu entwickelnde System muss hochverfügbar sein, aus diesem Grund kommt eine zentrale Alert-Meldestelle, welche ausfallen könnte, nicht in Frage. Jedes System muss in der Lage sein, eine Meldung zu versenden sowie eigenständig zu eruieren, ob eine Meldung versendet werden soll und kann. Die einzelnen Systeme müssen sich also gegenseitig sperren können, was durch ein Mutex-Verfahren[24] realisiert wird. Ein einzelnes System muss also alle anderen Systeme mit einem *Lock*[25] versehen können. Dabei wird basierend auf dem ausgefallenen Dienst, dem System und der Ausfallzeit auf jedem System ein *Semaphor*[26] erstellt, welcher dieses Locking steuert. Durch die Zeit des Fehlers sowie eines konfigurierten Timeouts kann so jedes System feststellen, ob ein Fehler gemeldet werden soll oder nicht.

Ein möglicher Ablauf einer Überwachung und Alert-Übermittlung ist in Abbildung 3.3 zu sehen. Bei dieser Visualisierung wird sowohl die PUSH- wie auch die POLL-Methode gezeigt. Bei einem POLL wird jedes mal beim Auftreten eines Fehlers jedes andere System angefragt, wann und ob der Fehler schon versandt worden ist. Jedes System gibt bei der Polling-Methode eine Semaphor zurück, anhand welcher dies dann geprüft werden kann. Rein rechnerisch kann die Anzahl Verbindungen, welche bei der POLL-Methode anfallen, durch

$$C = e \times (n - 1) \tag{3.6}$$

berechnet werden. Wobei C die Anzahl an Verbindungen, e die Anzahl der gleichen Fehler und n die Anzahl Server beschreibt. Diese Formel kann auch für eine Standard-Implementation der PUSH-Methode verwendet werden, denn auch bei dieser muss bei jedem auftreten eines Fehlers eine Semaphor an jedes System gesendet werden.

Um die optimalste Anzahl an Verbindungen zu erlangen, muss die Anzahl an Fehlern e in der obigen Formel 3.6 reduziert oder gar weggelassen werden. Dieser Ansatz wurde in der PUSH-Methode in Abbildung 3.3 verwendet: Die Semaphor wird nur dann gesendet, wenn auch ein Alert verschickt wird. Das setzt voraus, dass jedes System alle Meldungen empfängt und intern speichert, denn nur so können die Systeme die Semaphor für das Locking der Alerts nutzen. Semaphoren, welche durch die angegebene Zeiteinheit das

[24] Ein Mutex, englisch für **mut**ual **ex**clusion (wechselseitiger Ausschluss), verhindert, das nebenläufige Prozesse oder Threads gleichzeitig auf gleiche Datenstrukturen zugreifen und diese verändern.

[25] *Locking* bedeutet *Sperren*, also das beabsichtigte verhindern der Ausführung einer Aktion, Funktion, Dateizugriff, etc.

[26] Ein Semaphor beschreibt in der Informatiok eine Datenstruktur zur Steuerung des ausschliessenden Zugriffes. Ein Semaphor beinhaltet alle Daten, welche genutzt werden, um den Ausführungszeitpunkt zu bestimmen.

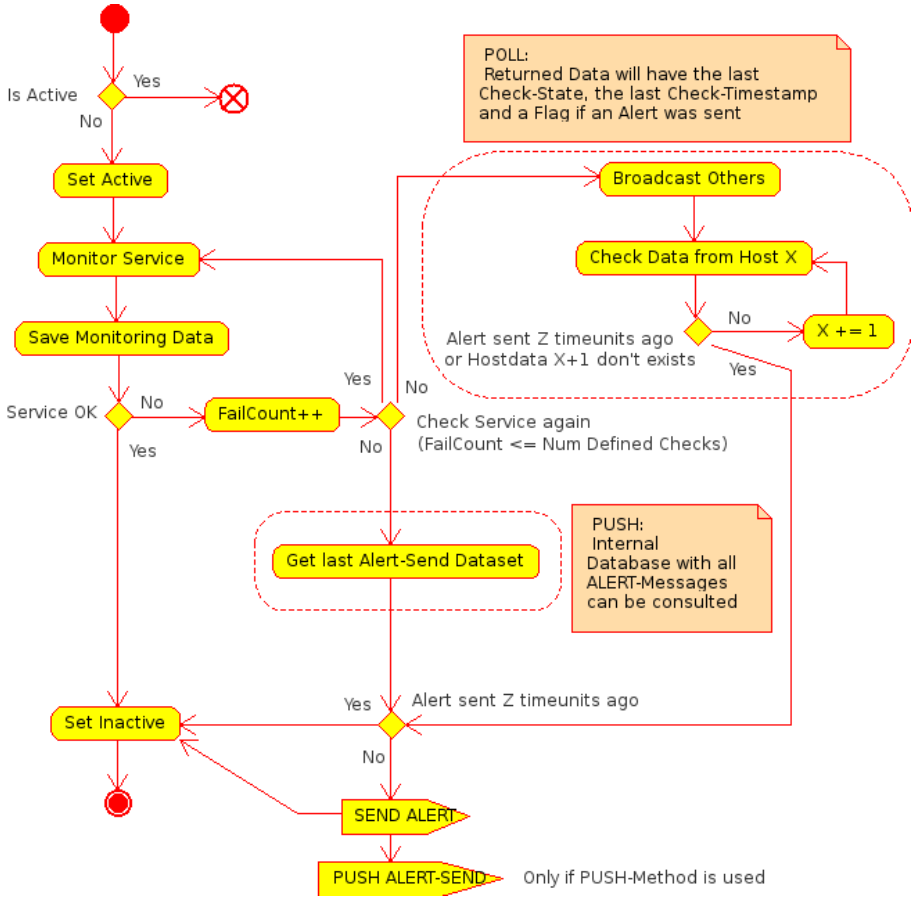

Abbildung 3.3: Eine Überwachung wird nur dann gestartet, wenn aktuell keine aktiv am laufen ist. Schlägt ein Test fehl, kann dieser erneut durchgeführt werden. Wenn alle Tests fehlgeschlagen sind, werden entweder alle anderen Systeme angefragt (POLL) und anhand deren Daten entschieden ob ein Meldung versendet werden muss oder nicht oder es werden die Internen Daten des letzten PUSH-Requests dazu verwendet.

Alerting nicht mehr blockieren, können aus dem Speicher entfernt werden, so dass neue Fehler wieder gemeldet werden können. Unter Berücksichtigung dieser Definition kann die Verbindungsanzahl durch

$$C = (n - 1) \tag{3.7}$$

berechnet werden.

Ist ein System, durch einen Netzwerkausfall oder ähnliches, komplett isoliert[27] von allen anderen Überwachungssystemen, kann ein Semaphor weder angefragt noch empfangen werden. Ist dies der Fall, und das Alerting ist nicht durch einen Lock gesperrt,

[27] Ein System kann komplett isoliert sein durch fehlende oder falsche Routen, ausgesteckte oder defekte Netzwerkkabel, defekte Netzwerkkarten, uvm.

wird der Fehler von dem System grundsätzlich gemeldet. In den meisten Fällen führt dies aber zu keinem doppelten oder mehrfachen Alert-Meldungs aufkommen, denn durch den Netzwerkfehler sind meistens auch die Alert-Nachrichten-Server (SMTP, SMS, etc.) nicht erreichbar. Durch die vernetzte Struktur wird das isolierte System aber von den anderen Servern erkannt und gemeldet.

Durch dieses Mutex-Verfahren kann bei den Alert-Meldungen komplett auf ein kompliziertes Transaktions-System verzichtet werden. Die Kombination des Push von Semaphoren und dem Locking reicht in den meisten Fällen dazu aus, ein mehrfaches versenden von Alert-Meldungen zu unterbinden. Ausgeschlossen werden kann dies jedoch nicht, da der Empfangszeiptunkt der Semaphor nicht garantiert werden kann. Es besteht bei dieser Variante also die Möglichkeit, dass eine Semaphor zum Zeitpunkt X gesendet und von einem System erst zum Zeitpunkt $X + 2$ empfangen wird. Prüft nun das zweite System zum Zeitpunkt $X + 1$ ob das senden erlaubt ist, weiss es nichts vom Sendeprozess des ersten Systems und sendet ebenfalls einen Alert. Durch eine Wartezeit zwischen dem Senden der Semaphore und dem Senden des Alerts sowie einer erneuten Prüfen, kann dieses Problem umgangen werden. Wird bei der zweiten Prüfung eine Semaphore gefunden, also ein Sendeprozess eines anderen Systems, erhällt jeweils das System mit dem kleineren Zeitstempel, also das erste System, den Vortritt zum Senden des Alerts.

3.6 Techniken zur Nachrichtenübermittlung

Zur Übermittlung von Nachrichten, sei dies durch einen Push- oder einen Pop-Request[28], gibt es nicht nur verschiedene Techniken sondern auch verschiedene Technologien. Zur Übermittlung von Überwachungsdaten haben sich dabei nachfolgende Standards etabliert, welche bei der zu implementierenden Software auch eingesetzt werden sollen.

3.6.1 SNMP

Das *Simple Network Management Protocol* (SNMP) ist ein standardisiertes Netzwerkprotokoll zur Überwachung und Steuerung von Netzwerkkomponenten. Das Protokoll wurde so festgelegt, dass prinzipiell jedes netzwerkfähige Gerät in die Überwachung aufgenommen werden kann. Dabei wird nicht nur der Aufbau der Datenpakete definiert, sondern ebenso der komplette Kommunikationsablauf. Um die Grundlast auf dem Netzwerk so gering wie möglich zu halten, wird das Verbindungslose Protokoll UDP genutzt.

[28] Ein Push-Request ist eine unaufgeforderte Zusendung von Daten; ein Pop-Request hingegen ist das aktive Anfragen von Daten

SNMP kennt insgesamt sechs verschiedene Paketarten. Die drei verschiedenen GET-Pakete[29] werden jeweils von einem Manager[30] an einen Agenten gesendet, worauf dieser mit einem RESPONSE-Paket antwortet. Das SET-Paket wird genutzt um Parameter auf einem Agenten zu verändern. Das TRAP-Paket schlussendlich ist ein einfacher Push-Request. Es wird also unaufgefordert ein Paket von einem Agenten an einen Manager gesendet. Traps werden dazu genutzt, um einen Manager darüber zu informieren, dass ein Ereignis aufgetreten ist. Der Agent kann dabei nicht feststellen, ob das Paket beim Manager angekommen und verarbeitet worden ist, denn ein TRAP-Paket kann nicht durch ein RESPONSE-Paket bestätigt werden.

SNMP definiert zwar den Aufbau der Datenpakete (siehe Anhang 5.5), nicht jedoch die Daten selbst, welche übertragen werden. Die Daten und Werte, welche von einem Gerät ausgelesen werden können, werden als „Managed Object“ übertragen. Diese MOs sind durch eine `Management Information Base` (MIB)[31] definiert. Die Basis jedes MIB ist der `Object Identifier` (OID)[32] sowie die Definition der unterschiedlichen Datenfelder. Diese Datenfelder weisen nicht nur einen Namen auf, sondern unterliegen auch einem Datentyp. Datentypen können einfache Zahlen oder Texte sein, jedoch auch vordefinierte Muster wie zum Beispiel eine IPv4-Adresse (siehe Codeblock 5 im Anhang).

Aktuell sind verschiedene Versionen von SNMP verfügbar und im Einsatz. SNMPv1 wurde 1988 definiert und setzte den Fokus nicht gross auf die Datensicherheit. Aus diesem Grund wurden verschiedene SNMPv2 Versionen definiert, welche alle unterschiedliche Sicherheits- und Datenkonzepte aufweisen. Die am meisten verwendete Version ist SNMPv2-community (SNMPv2c). Die Sicherheit basiert bei SNMPv2c auf so genannten „Communities“, bei welchen definiert werden kann, dass z.B. bei `PUBLIC` nur gelesen, bei `PRIVATE` jedoch zusätzlich auch geschrieben werden darf. Eine Sicherheitsfunktion wie ein Passwort gibt es dabei nicht. Es kann zwar ein komplizierter Community-Name definiert werden, da SNMPv2c aber nicht verschlüsselt wird, bringt auch das keinen wirklichen Schutz.

[29] SNMP kennt ein GET-Paket zum Anfragen von Daten, das GETNEXT-Paket um einen nachfolgenden Datensatz z.B. aus einer Tabelle aus zu lesen sowie das GETBULK-Paket zum Abrufen von mehreren Datensätzen

[30] Bei SNMP wird der Server mit Manager und der Client mit Agent beschrieben.

[31] Die MIB, definiert in RFC-1155[RM90] (siehe auch RFC-4293[Rou06], RFC-4022[Rag05], RFC-4113[FF05]), beinhaltet keine Daten sondern den Aufbau eines „Managed Objects“, welches verwendet wird um Daten zu versenden/empfangen.

[32] Eine OID sollte eine Weltweit eindeutige Identifikation sein, um Verwechslungen zu vermeiden. Firmen können bei der IANA (Internet Assigned Numbers Authority, `http://www.iana.org/`) kostenlos eigene MIB-Module unterhalb „iso.org.dod.internet.private.enterprise“ beantragen (`http://pen.iana.org/pen/PenApplication.page`). Unterhalb dieses Zweiges können anschliessend eigene OIDs definiert werden.

Die Versionen SNMPv1 bis SNMPv2c bieten also weder eine gute Verschlüsselung noch eine sichere Möglichkeiten zur Authentifizierung. Aus diesem Grund wurde im Dezember 2002 die Spezifikation von SNMPv3[33] veröffentlicht, welche Benutzer- und Ansichts basierende Zugriffsbeschränkungen sowie andere Sicherheitsmechanismen beinhaltet. Gerade wegen diesen neuen Sicherheitsfunktionen, wie zum Beispiel der Schlüsselverwaltung, ist SNMPv3 noch nicht sehr verbreitet.

Fazit: SNMP

Trotz gravierender sicherheits bezogener Mängel der Versionen SNMPv1 bis SNMPv2c ist SNMP dennoch eine schnelle und einfache sowie gut erweiterbare Technologie zur Abfrage und Übermittlung von einfachen Datensätzen. Sofern die Sicherheit keine grosse Rolle spielt, können diese SNMP-Versionen ohne Bedenken eingesetzt werden. Zur Übertragung von kritischen Daten muss entweder SNMPv3 in einer verschlüsselten Variante oder SNMPv2c/SNMPv3 über eine verschlüsselte Verbindung respektive über ein komplett abgeschirmtes Management-Netzwerk verwendet werden.

SNMP kann genutzt werden, wenn Daten von einem System auf ein anderes übertragen werden sollen. So kann SNMP zum Beispiel gut für eine Leistungsüberwachung genutzt werden, nicht jedoch für die Funktionsprüfung eines Webservices oder anderen Dienstes wie SMTP, POP3, etc.

3.6.2 IPMI

Der *Intelligent Platform Management Interface* (IPMI) Standard wurde von verschiedenen Hardware-Herstellern entwickelt zur einfachen Wartung und Verwaltung von Hardware. Die Betriebssystem unabhängigen Schnittstellen erlauben es zudem, automatische Berichte über allfällige Probleme zu versenden. Sie können entweder über eine serielle Schnittstelle oder auch über das lokale Netzwerk genutzt werden. IPMI funktioniert sowohl während dem laufenden Betrieb, jedoch auch wenn dieses im Standby oder ausgeschaltet ist. Die einzige Voraussetzung ist eine Betriebsspannung auf den Komponenten.

Das Herzstück von einem IPMI Gerät ist der `Baseboard Management Controller` (BMC)[34]. Dieser Controller ist ein spezieller Mikrocontroller, an welchen alle dafür aus-

[33] SNMPv3 wird in folgenden RFCs definiert: RFC-3410[CMPS02], RFC-3411[HPW02], RFC-3412[CHPW02], RFC-3413[LMS02], RFC-3414[BW02], RFC-3415[WPM02], RFC-3416[Pre02c], RFC-3417[Pre02b], RFC-3418[Pre02a]

[34] Auf den meisten heutigen Server-Boards ist bereits ein BMC verbaut, was die Überwachung mittels IPMI sehr kostengünstig und einfach macht

gelegte Hardware angeschlossen werden kann[35]. Wird mit dem BMC über das Netzwerk kommuniziert, kann optional eine Verschlüsselung verwendet werden. Bei einer seriellen Verbindung wird grundsätzlich angenommen, dass die Verbindung sicher ist. Für den Verbindungsaufbau wird jedoch in jedem Fall ein Passwort benötigt.

Fazit: IPMI

IPMI ist grundsätzlich ein Management-Tool für Server und nicht ein Protokoll zur Datenübertragung. Beispielsweise kann IPMI dazu genutzt werden, einen Server aus der Ferne zu analysieren, neu zu starten, den Startvorgang zu beobachten, etc. Dafür muss das System aber über die notwendige Hardware verfügen und entsprechend konfiguriert sein.

Durch verschiedene Tools wie *IPMItool*[IPM07] oder *OpenIPMI*[Ope10a] wird die Administration sowie der Zugriff auf ein IPMI-Systeme stark vereinfacht und kann auch automatisiert werden.

Bei der zu implementierenden Software kann angedacht werden, dass IPMI als zusätzliche Datenabfrage zur Auswahl steht.

3.6.3 Andere Formate

Je nach Art der Prüfung, sind eventuell andere Formate notwendig. Soll ein Service auf einem Webserver auf Funktionalität geprüft werden, kann nicht SNMP verwendet werden. Die Anfrage an den Dienst muss mittels HTTP über TCP gemacht und die Antwort anschliessend entsprechend ausgewertet werden. Das selbe gilt für DNS-Prüfungen, Mailserver-Tests und andere.

3.6.4 Fazit: Andere Formate

Die Art der Datenübermittlung darf bei dem zu entwickelnden System nicht eingeschränkt werden. Das System soll durch Plugins nicht nur erweiterbar sein, sondern es sollen auch verschiedene Dienste und Services geprüft werden können. Die Art und das Format der Daten bei einer Überwachung darf nicht eingeschränkt werden.

[35] Die Hardware muss über einen SMBus verfügen. Dies ist ein zweiadriger Bus mit maximal 100 kbit/s und basiert auf dem I^2C-Serialbus von Philips und ist abwärtskompatibel zu diesem.

3.7 Techniken zur Alert-Übermittlung

Je nach Schweregrad der Meldung muss eine solche schnell und umgehend einer verantwortlichen Person oder Stelle zugestellt werden. Dies setzt voraus, dass eine Nachricht über verschiedene Kanäle übermittelt werden kann, welche ebenfalls unterschiedlich priorisiert werden können.

3.7.1 EMail

EMail ist ein relativ unsicheres Medium zur Alert-Übermittlung. Zum einen muss das System eine Verbindung mit einem EMail-Server besitzen, zum anderen muss der Empfänger auch vor dem Computer sitzen und regelmässig das entsprechende Postfach prüfen.

Ein Vorteil von EMail ist sicherlich, dass alle Daten von einem Test direkt übermittelt werden können. Voraussetzung ist, dass der EMail-Server funktioniert und eine Verbindung mit diesem hergestellt werden kann. Der Fehler darf somit weder etwas mit dem Netzwerk noch mit der Internetverbindung oder dem EMail-Server zu tun haben.

Eine Möglichkeit wäre, das EMail direkt vom System aus an den Empfänger-Mailserver zu senden, ohne Umweg über einen internen/externen Mailserver. Durch die heutzutage eingesetzten Spam- und Virenfilter, treten dabei wahrscheinlich mehr Probleme auf als verhindert werden. Viele Mailserver blocken den ersten Zustellversuch[36], lassen unbekannte oder dynamische Absender erst gar nicht durch[37], verlangsamen absichtlich den Datenversand[38] oder blocken Systeme, welche keine Mailserver sind, komplett[39].

3.7.2 Fazit: EMail

Die Meldungszustellung über EMail sollte grundsätzlich nur für Informative Zwecke verwendet werden und nicht für hoch kritische Nachrichten. EMail ist nicht nur langsam, sondern es kann auch nicht gewährleistet werden, dass eine Meldung wirklich zugestellt wird - vorallem nicht in einem angemessenen Zeitrahmen.

[36] Beim Graylisting wird jeweils ein Zustellversuch geloggt und das erste mal geblockt. Erst der zweite Versuch vom gleichen Absender, an den gleichen Empfänger über den gleichen Server wird dann erlaubt. Somit kommt ein EMail vielfach erst mit einer Verzögerung von einigen Minuten oder Stunden beim Empfänger an.

[37] Durch Realtime-Blacklists (RBL) werden dynamische Absender und bekannte Spamnetzwerke gesperrt

[38] Durch Tarpitting werden Verbindungen künstlich verlangsamt um zu schauen ob die Gegenstelle ein richtiger Mailserver ist und kein Spamhost, welcher keine Zeit hat da er viele Nachrichten versenden muss

[39] Teilweise werden Verbindungen zu den Absendern oder den Reverse-DNS Einträgen aufgebaut um zu prüfen ob dort ein richtiger Mailserver ist

3.7.3 SMS-Nachricht

Das Senden und Empfangen von SMS-Nachrichten ist eine schnelle und einfache Art der Nachrichtenzustellung. Eine SMS ist beschränkt in der Anzahl Zeichen, welche versendet werden können. Somit könnte beim Auftauchen eines Fehlers lediglich eine kurze Beschreibung oder eine Zusammenfassung versendet werden.

Die Nachrichtenzustellung per SMS hat drei gravierende Nachteile:

- Ein System muss entweder mit einer GSM-Komponente ausgestattet werden, welche den Versand der SMS übernimmt oder es muss ein Online-Dienst zum Versenden in Anspruch genommen werden.
- Die Zustellung einer SMS wird von den Mobile-Com-Firmen nicht garantiert, denn die Sprachdienste sind auf dem GSM-Netzwerk höher priorisiert als Datendienste, es sei denn, man hat spezielle Verträge, welche aber nur Behörden und Notfalldienste bekommen [Tan03].
- Ist der Empfänger nicht erreichbar, kann nicht gewährleistet werden, dass die Nachricht wirklich zugestellt wird oder wurde [Tan03].

Weiter zu bedenken ist, dass bei vielen Dienstleistern das Versenden einer SMS Kosten verursacht. Es kann passieren, dass bei einem Fehler oder einer falsch konfigurierten Regel eine Unmenge an Nachrichten versendet werden, welche somit hohe Kosten verursachen.

Fazit: SMS-Nachrichten

Jedes System mit einer GSM-Komponente sowie mit einer gültigen SIM-Karte und einem Vertrag aus zu statten, würde nicht nur in einer kleinen sondern auch in einer grossen Konfiguration den Kostenrahmen sprengen. Eine Möglichkeit wäre, pro Netzwerk nur einige Systeme mit einer GSM-Komponente aus zu statten. Zusätzlich sollte auch die Einbeziehung von einem oder mehreren Online-Services für den SMS-Versand genutzt werden können.

Die Gewährleistung, dass eine Nachricht wirklich zugestellt wird, kann nicht gegeben werden[40]. SMS sollte also immer in Kombination mit einer anderen Variante verwendet werden: Die Verantwortliche Person wird per SMS und per EMail, welches alle gesammelten Daten beinhaltet, über den Fehler informiert.

[40] Bei SMS kann nur durch eine Statusabfrage geprüft werden, ob eine Nachricht dem Empfänger zugestellt worden ist oder nicht [Tan03].

3.7.4 Paging

Paging, oder auch Funkmeldeempfänger (FME), wird vielfach zur Nachrichtenübermittlung und Alarmierung eingesetzt. Der grosse Vorteil eines Pagers gegenüber einem GSM-Gerät ist, dass ein Pager durch die unterschiedliche Technik nicht nur in abgelegenen Bereichen[41], in welchen ein Mobiltelefon keinen Empfang hat, funktioniert, sondern auch, dass die Zustellung einer Paging-Nachricht prinzipiell gewährleistet werden kann. Aus diesem Grund setzen immer noch viele Notfalldienste wie Feuerwehr und Rettungsdienst auf diese Technik und nicht auf GSM.

Ein FME kann prinzipiell mit einem Funkgerät verglichen werden, welches dauernd auf Empfang ist. Unterliegen die empfangenen Daten einem Muster, also einer Reihe von Tönen, werden alle nachfolgenden Daten weiterverarbeitet und angezeigt. Die Digitalen Melde Empfänger (DME und POCSAC) arbeiten auf einer höheren Frequenz als die FME[42]. Die DME-Geräte können zudem in verschiedenen Ausbaustufen genutzt werden. Die einfachste Stufe, die DME-I Geräte, besitzen lediglich vordefinierte Texte, welche mittels einer gesendeten Bitfolge aufgerufen werden können. Die zweite Ausbaustufe, die DME-II Geräte, sind zudem noch in der Lage, Freitextnachrichten (ähnlich einer SMS) zu empfangen. Die höchste Ausbaustufe bieten DME-III Geräte, welche den empfangenen Text anhand eines integrierten Lexikons in Sprache übersetzen und wiedergeben.

Die kommenden Generationen werden wohl über das TETRA-Netzwerk agieren, welches ähnlich wie das GSM-Netzwerk auf Zellen basiert und somit nicht nur eine „Gesprächsweitergabe“ sondern auch eine Lokalisierung erlaubt[43]. Vorteil der TETRA-basierenden Geräte ist die Möglichkeit, auf eine Meldung reagieren zu können. Dies ist bei den aktuellen FME-, DME- und POCSAG-Geräten nur in Kombination mit einem weiteren Kommunikationskanal wie GSM möglich. Der Nachteil der TETRA-Netzwerke ist aber der selbe wie bei GSM: Durch die Zellen können wie bei GSM viele kleine Funklöcher entstehen[44].

[41] Die Abdeckung der Schweiz durch die SWISSPHONE liegt beim Paging bei ca. 99%, wie auch die Netzabdeckung von GSM durch die Swisscom, Orange oder Sunrise. Bei GSM gibt es jedoch viele kleine Inseln, welche auch in urbanen Gegenden zu einer schwachen oder gar keiner Abdeckung führen können.

[42] Ein FME ist rein Analog, es werden also Töne übertragen welche ausgewertet werden. Ein DME hingegen kann auf dem 2-Meter Band arbeiten, welches Analoge Signale nicht mehr geeignet ist. Die POCSAG-Geräte arbeiten in einer noch höheren Frequenz

[43] Ein Alert könnte somit derjenigen Person zugestellt werden, welche geographisch gesehen am nächsten bei dem fehlerhaften System ist.

[44] Die Problematik der Funklöcher müssen aktuell diverse deutsche Stellen und Städte erfahren, welche eine Umstellung auf den POCSAC-Standard vollziehen ohne diesen ausführlich getestet zu haben. Ein Beispiel hierzu ist die Stadt München: http://www.heise.de/security/meldung/Muenchen-mit-grossen-BOS-Digitalfunk-Loechern-1148126.html

Fazit: Paging

Die Nachrichtenübermittlung über ein DME- oder POCSAG-Gerät ist eine schnelle, einfache und vor allem zuverlässige Möglichkeit einen Alarm zu versenden. Obwohl die Netzabdeckung des Paging-Netzwerkes zahlenmässig ungefähr gleich ist wie diejenige des GSM-Netzwerkes, gibt es dennoch weniger Funklöcher, da das Paging-Netzwerk nicht auf verschiedene Frequenzen und unterschiedliche Zellen aufgeteilt ist. Das Paging-Netzwerk ist eine grosse Fläche. Befindet man sich innerhalb dieser Fläche, kann man eine Nachricht empfangen (ähnlich wie das ein Funkamateur macht).

Ein grosser Nachteil einer Paging-Nachricht ist jedoch die „Nicht-Überprüfbarkeit“ der Zustellung. Auch das eigentliche Versenden einer Pager-Nachricht kann sich als schwer erweisen, denn diese werden in den meisten Fällen direkt über das Internet zum Netz-Anbieter gesendet und dann von diesem weiter verschickt. Ist die Internetanbindung also nicht vorhanden, kann kein Alarm ausgelöst werden.

Zudem kann auch die Kostenfalle zu einem Problem werden, wie schon bei SMS beschrieben.

3.7.5 Andere Möglichkeiten

Neben den vorgestellten Möglichkeiten zur Übermittlung von Meldungen, gibt es noch weitere Techniken für die gleiche Angelegenheit. So können zum Beispiel alle Meldungen mittels einem SNMP-Trap an eine oder mehrere Stellen gesendet werden, welche diese dann auswerten und anzeigen. Wenn jedoch mehr Daten versendet werden sollen, könnte dies auch über eine einfache XML-Schnittstelle und HTTP erledigt werden oder über eine einfache RS232-Schnittstelle (COM-Port), an welcher ein Display, eine Alarmzentrale, etc. angeschlossen ist.

3.7.6 Fazit: Techniken zur Alert Übermittlung

Die Alarmierung ist in den meisten Fällen abhängig von einer funktionierenden Internet-Verbindung. Sei dies für eine EMail-Nachricht oder um den Paging-/SMS-Anbieter zu erreichen, welcher Nachrichten über einen Webservice empfangen und versenden kann. Verschiedene Techniken wie SMS oder Paging würden es auch erlauben, externe Hardware für den Versand zu nutzen. Ob diese Hardware nun an jedem Überwachungssystem angeschlossen ist oder nur an einer zentralen Stelle im Netzwerk, darf bei der zu implementierenden Lösung keine Rolle spielen.

Eine Alarmierungs-Technik sollte grundsätzlich über eine Schnittstelle angesprochen und so erweitert werden können. Dadurch können verschiedene Möglichkeiten unter-

schiedlich genutzt werden. Sei dies nun eben, dass ein SMS mittels einem zentralen SMS-Gateway, einer RS232-Schnittstelle oder einem Webservice versendet werden soll. Bei der Überwachungssoftware sollte prinzipiell nur gesagt werden, dass bei einem Alarm vom Typ XY, ein SMS versendet werden soll sowie zusätzlich ein EMail mit allen Daten. Die komplette Priorisierung, was genau bei einem Alarm vom Typ XY gemacht werden soll, wird in der jeweiligen Alarmierungs-Erweiterung definiert.

4 Praktische Umsetzung

Die praktische Ausarbeitung stellt ein *Proof of Concept* dar, also ein Machbarkeitsnachweis der im theoretischen Teil hergeleiteten Lösungsansätze. Die Schwerpunkte sind die parallele Ausführung verschiedener Aktionen, die dynamische Erweiterung durch Module sowie die Ausprogrammierung und Herleitung der Theorien für das Host-Discovery, die Anordnung der Systeme innerhalb eines Hypercubes, die Alert-Übermittlung und Prüfung sowie die Performance im Allgemeinen.

Als Modell für die Programmierung und die generelle Entwicklung des Überwachungs-Systems wird eine Mischform aus Experimentellem- sowie Evolutionärem-Protoyping und einer spiralförmigen Entwicklung gewählt. Dies bedeutet, dass Funktionen mittels der Software ausprobiert, verbessert und analysiert, jedoch auch wieder entfernt oder komplett anders gestaltet werden können. Das kann in Zusammenarbeit mit Endanwendern oder auf Grund von Tests und Analysen geschehen. Die Verbesserung der Performance sowie des Funktionsumfanges wird jedoch eher in einem spiralförmigen Prozess ablaufen. Die Grundfunktionen des Systems werden also relativ gering ausfallen und sind unter Umständen auch noch nicht optimal implementiert. In verschiedenen Zyklen werden je nach Bedarf neue Funktionen und Schnittstellen implementiert oder die Performance von solchen verbessert und angepasst.

Durch diese Vorgehensweise wird das System nicht nur anwenderfreundlich, sondern auch die Performance wird nur an den Stellen verbessert, an welchen es überhaupt sinnvoll ist. Der Weg von der hier entwickelten Technologie-Demonstration zu einem fertigen Produkt ist anschliessend nur ein kleiner.

Das Grundsystem muss schon von der Basis aus sehr performant laufen, es sollte so wenig Rechenleistung und Speicher wie möglich benötigen. Aus diesem Grund, und um nicht auf die Vorteile der Objektorientiertheit zu verzichten, wird als Programmiersprache Vala[val10] eingesetzt und keine Interpreter-/Skript-Sprache oder C#/Java welche eine VirtualMachine als Laufzeitumgebung voraussetzen. Vala ist eine an C# angelehnte, objektorientierte Programmiersprache, deren Compiler den Vala-Code zuerst in reinen C-Code umwandelt und diesen anschliessend kompiliert. Auf anderen Betriebssystemen und Architekturen kann entweder der erzeugte C-Code oder auch direkt der Vala-Code

kompiliert und anschliessend verwendet werden. Die einzige Voraussetzung ist, nebst der Einhaltung gewisser Paradigmen, dass es Vala, die GLib[gli10] sowie Gee[gee10] und GIO[gio10] für die entsprechende Plattform und Architektur gibt.

4.1 Grundsystem

Das Grundsystem, siehe Abbildung 4.1, stellt die zentrale Plattform des Überwachungs-Systems dar. Die zentrale Einheit bildet dabei das `MainSystem`, welches nicht nur globale Funktionen zur Verfügung stellt, sondern auch als Schnittstelle zwischen den einzelnen Bereichen fungiert. Um diesen System-Kernel angeordnet befinden sich die Bereiche zur Datenübertragung (Netzwerk-und Bus-Technologien), zur Überwachung von Diensten und Systemen (Überwachungsmodule), zur Datenspeicherung, für das Melden von Vorkommnissen (Alerting) sowie Schnittstellen für die Datenanzeige auf Remote-Systemen (Remote-API). Diese verschiedenen Bereiche werden als Module implementiert und nur bei Bedarf geladen. Die Module können sowohl in binärer Form als auch als LUA-Skript[1] vorliegen - wobei die LUA-basierenden Module nicht im Rahmen der Thesis implementiert werden.

4.1.1 System-Kernel

Die zentrale Schnittstelle, der System-Kernel, ist grundsätzlich verantwortlich für die komplette Thread-Verwaltung sowie für die Kommunikation zwischen den Threads und den Modulen.

Nach dem Start des Programms wird eine Instanz des System-Kernels angelegt und die verschiedenen Modul-Container geladen. Sind alle Modul-Container erfolgreich geladen, wird zuerst der `MainSystem`-Thread, anschliessend der `Monitoring`-Thread und zum Schluss der `MainListener`-Thread gestartet. Die verschiedenen Threads werden in den nachfolgenden Kapiteln beschrieben.

Durch einen Zähler, welcher beim Erstellen einer Modul-Container Instanz inkrementiert wird, sowie einem Signal, welches der Conatiner sendet, wenn alles geladen ist, kann geprüft werden, ob alle Container geladen worden sind. Vor der Initialisierung wird also ein Zähler inkrementiert sowie eine Funktion des System-Kernels an das Signal gebunden (Codeblock 4.1). Konnten alle Module erfolgreich geladen werden, wird vom Container

[1] LUA[lua10] ist eine imperative Skriptsprache welche sich einfach erweitern und einfach in C-Programme einbinden lässt. Vorteil von LUA-Skripten ist die Plattformunabhängigkeit, sowie die Möglichkeit, einfachen Code zu schreiben welcher mit dem vorhandenen System über eine definierte Schnittstelle kommunizieren kann.

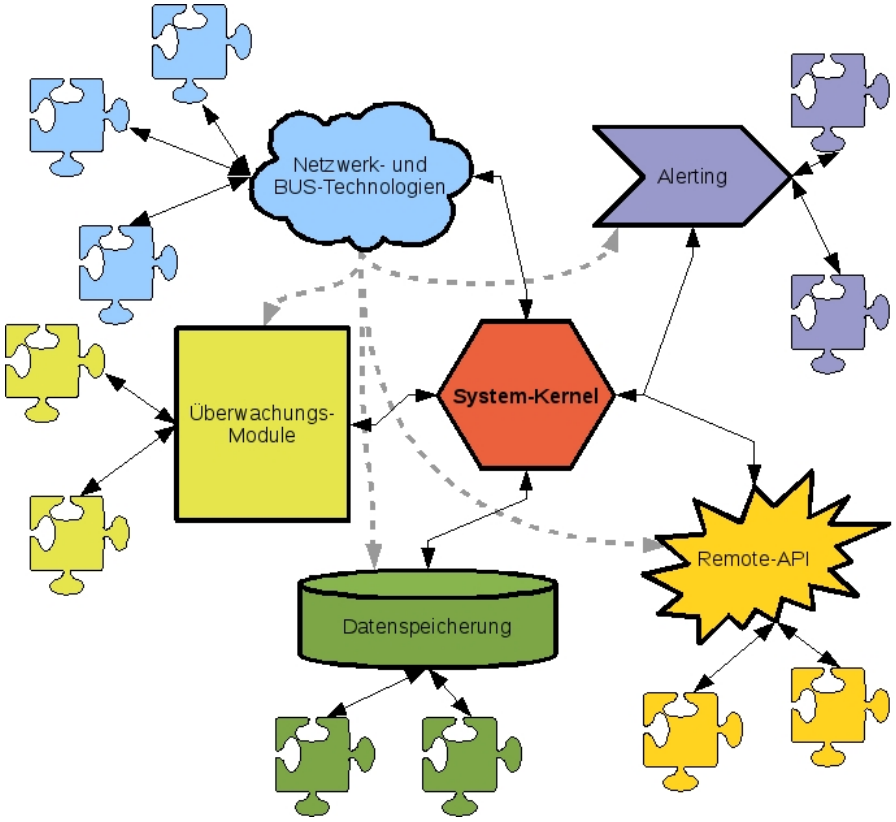

Abbildung 4.1: Das Überwachungs-System bietet rund um den zentralen System-Kernel verschiedene Funktionen für unterschiedliche Bereiche an. Diese verschiedenen Bereiche stellen jeweils Container für Module dar, welche sowohl in binärer Form als auch als LUA-Skript vorliegen können. Eine Modul-Instanz kann beim System-Kernel eine Instanz eines anderen Moduls anfragen und dieses direkt nutzen. Dieser Prozess ist mit den „Netzwerk- und BUS-Technologien"-Modulen schematisch dargestellt.

das Signal aufgerufen und dadurch die vorgängig daran gebundene Funktion. Diese Funktion dekrementiert den Zähler und führt weitere Ladeaktionen aus wenn dieser wieder bei 0 angelangt ist (Codeblock 4.2).

Der Thread des `MainSystem` dient vor allem dazu, eine Asynchrone MessageQueue ab zu arbeiten. Nachrichten in dieser Queue können von unterschiedlichen Stellen eingespiesen werden. Eine Nachricht besteht immer aus einem Kommando und optionalen Daten (siehe Codeblock 5.3). Die Nachrichten werden nach dem FIFO-Prinzip (first in first out) abgearbeitet, sie werden also der Reihe nach ausgelesen und das darin enthaltene Kommando ausgeführt. Sind keine Daten mehr in der `MessageQueue`, wird der Vorgang für 100ms pausiert. In einer zukünftigen Version kann angedacht werden, anstelle der Pausierung für 100ms diesen komplett zu stoppen und beim einspeisen einer Nachricht ein Signal ab zu setzen, welches den Thread wieder aufweckt.

Eine weitere Aufgabe des `MainSystem`-Threads sind diverse Aufräumarbeiten. Bei einem Host-Discovery wird beispielsweise eine Objekt-Liste mit verschiedenen Discovery-Prozessen (siehe Kapitel 4.1.7) angelegt. Ist ein Prozess abgeschlossen, muss dieser beendet und wieder freigegeben werden.

Globale Kernel-Funktionen

Zur Kommunikation mit den unterschiedlichen Modulen und den verschiedenen Modul-Containern stehen im System-Kernel einige öffentliche Funktionen zur Verfügung:

exit (): Beendet alle Threads und das Programm

getComm (string name): Erzeugt ein neues Kommunikations-Modul (ICommModule) und gibt dieses zur Verwendung zurück.

getData (string name): Erzeugt ein neues Datenbank-Modul (IDataModule) und gibt dieses zur Verwendung zurück.

getAlert (string name): Erzeugt ein neues Alert-Modul (IAlertModule) und gibt dieses zur Verwendung zurück.

getRemote (string name): Erzeugt ein neues Remote-API-Modul (IRemoteModule) und gibt dieses zur Verwendung zurück.

getMonitor (string name): Erzeugt ein neues Monitoring-Modul (IMonitorModule) und gibt dieses zur Verwendung zurück.

logMonitoring (string plugin, string host, string[] result): Nimmt eine Meldung von einem Überwachungsmodul entgegen und speichert diese in allen konfigurierten Datenbanken ab.

sendAlert (string plugin, string host, int timeout, string subject, string short_message, string message, string[] attachment=null): Nimmt eine Fehlermeldung entgegen und sendet einen Alert wenn dies erforderlich ist. Basierend auf dem Plugin und dem Host kann geprüft werden, ob ein Alert schon versendet worden ist oder nicht. Die Parameter werden 1:1 an das entsprechende Alert-Modul übergeben. Je nach Art der Nachricht werden die einzelnen Daten dabei gekürzt oder erst gar nicht versendet.

getDiscoverServiceList (): Gibt eine Liste mit allen Diensten zurück, auf welche ein System geprüft werden soll, wenn dieses beim Auto-Discovery gefunden worden ist.

Die Liste besteht aus `Helper.NetService`-Datensätzen. Die `Helper.NetService`-Datenstruktur ist im Anhang 5.2 zu sehen.

setDiscoveredHosts (Helper.NetService[] list): Systeme, welche durch ein AutoDiscovery gefunden und untersucht worden sind, können über diese Funktion dem System mitgeteilt werden. Die Systeme müssen als `Helper.NetService` vorliegen.

getDataFile (string file_name): Gibt ein `File`-Objekt für eine Datei oder ein Verzeichnis innerhalb des „data“-Verzeichnisses zurück.

getPath (string[] name): Gibt ein Unterverzeichnis als String zurück. Die Unterverzeichnisse müssen als String-Array übergeben werden.

getFile (string[] path, string name): Gibt eine Datei als String zurück. Die Unterverzeichnisse müssen als String-Array übergeben werden.

Modul-Container / PluginRegistrar

Ein Modul-Container ist prinzipiell eine einfache typisierte Liste, welche ein binäres Modul laden und Instanzen von diesem zurückgeben kann. Diese typisierte Liste wird implementiert durch den `PluginRegistrar`. Beim Instanzieren muss das Verzeichnis angegeben werden, welches die Module beinhaltet, die geladen werden sollen. Existiert das angegebene Verzeichnis, werden alle Unterverzeichnisse der Reihe nach durchlaufen und in diesen nach der Datei `libplugin.so` gesucht. Existiert eine solche Datei, wird sie in den Speicher geladen und versucht über die Funktion `register_plugin(Module module, PluginRegistrar registrar)` das Plugin zu initialisieren. Diese Einstiegsfunktion muss das Plugin durch den Typen, den Pfad sowie einen Identifier im entsprechenden `PluginRegistrar` speichern. Wird keine solche Einstiegsfunktion gefunden, kann das plugin nicht geladen werden.

Bei der Initialisierung eines `PluginRegistrar` muss immer auch ein Interface angegeben werden, welches für die geladenen Module verwendet wird. Zusätzlich muss auch noch ein Pointer auf das `MainSystem` übergeben werden, so dass die Module und auch der Container Anfragen an den System-Kernel stellen können (siehe Codeblock 4.1).

Damit der `PluginRegistrar` andere Komponenten darüber informieren kann, dass nun alle Module geladen sind, wird nach dem Laden ein Signal abgesetzt. Auf dieses Signal können sich andere Komponenten Funktionen registrieren, welche anschliessend aufgerufen werden (siehe Codeblock 4.1 und 4.2). Durch diese Funktionalität stellt der System-Kernel fest, wann alle Module geladen sind und mit dem Laden von anderen Komponenten und Threads weitergemacht werden kann.

```
1: // Counter to check if all Container are loaded
2: registrars_loaded++;
3:
4: comm = new PluginRegistrar<ICommModule>("pnetwork", this);
5: comm.complete.connect(modulesLoaded);
6: comm.load();
```

Codeblock 4.1: Dieser Modul-Container lädt alle Module im Verzeichnis `pnetwork` und ruft nach dem erfolgreichen Laden die Funktion `modulesLoaded` (siehe Codeblock 4.2) aus dem MainSystem auf. Durch den `load()`-Aufruf wird der Ladevorgang gestartet.

```
1: private void modulesLoaded(string path) {
2:   registrars_loaded--;
3:
4:   // All modules are loaded
5:   if (registrars_loaded <= 0) {
6:     ...
7:   }
8: }
```

Codeblock 4.2: Eine Funktion, die an das `complete`-Signal des `PluginRegistrar` gebunden wird, muss über den Parameter `string path` verfügen. Dieser beinhaltet den Pfad der Module. Diese Implementierung zeigt, wie gewährleistet wird, dass erst nach dem Laden aller Container gewisse Funktionen ausgeführt werden.

Jedes Plugin, das geladen wird, registriert sich entweder über die Funktion `addPlugin(IMainModule plugin)` oder über `addInstance(Type t, string path, string identifier)` bei dem Modul-Container. Ein Plugin kann sich dadurch als „bereits instanziert“ oder als „noch zu instanzieren“ registrieren. Durch die Festlegung, dass ein Überwachungssystem so wenig Resourcen wie möglich belegen darf, sollten bereits instanzierte Plugins so wenig wie möglich genutzt werden, da diese beim Systemstart komplett in den Speicher geladen werden und dort verbleiben (während dem die anderen geladen und anschliessend wieder freigegeben werden). Zudem sollte ein solches Plugin als Singelton implementiert werden, denn es kann nur einmal im System bestehen.

Plugins, welche über die `addInstance(...)`-Funktion registriert werden, können zu einem beliebigen Zeitpunkt instanziert werden. Dafür notwendig ist lediglich der Typ, welcher der Funktion übergeben wird. Damit dies jedoch funktioniert, muss beim Laden der Datei `libplugin.so` angegeben werden, dass dieses Modul speicherresistent sein muss, ansonsten wird der Speicher mit den Modul-Informationen beim verlassen der Funktion wieder freigegeben. Codeblock 4.3 zeigt einen schematischen Ablauf des Ladeprozesses. Bei Zeile 4 und 7 wird das Plugin über ein Modul geladen und auf Zeile 9 definiert, dass die Informationen speicherresistent gehalten werden müssen. Ansch-

liessend wird bei Zeile 12 geprüft, ob die Einstiegsfunktion `register_plugin` existiert. Ist dies nicht der Fall, wird abgebrochen. Auf den Zeilen 16 und 17 wird der Typ der Einstiegsfunktion definiert und diese anschliessend bei Zeile 18 aufgerufen.

```
1:  protected delegate Type RegisterPluginFunction(Module module,
2:                                                 PluginRegistrar registrar);
3:
4:  string plugin_file = Module.build_path(
5:      Path.build_filename(module_dir.get_path(), plugin_name),
6:      "plugin");
7:  Module module = Module.open(plugin_file,
8:                              ModuleFlags.BIND_LAZY);
9:  module.make_resident();
10:
11: void* function;
12: if (!module.symbol("register_plugin", out function)) {
13:   return;
14: }
15:
16: RegisterPluginFunction register_plugin =
17:     (RegisterPluginFunction)function;
18: register_plugin(module, this);
```

Codeblock 4.3: Ablauf zum Laden eines binären Modules. Zuerst wird das Modul geladen, speicherresident gehalten, auf die Einstiegsfunktion geprüft und diese, wenn gefunden, aufgerufen.

Basierend auf dem Identifier, welcher jedes Plugin pro Modul-Container eindeutig macht, kann der System-Kernel anschliessend eine Instanz eines Moduls anfragen. Ist das Plugin im angefragten ModulContainer vorhanden, wird basierend auf dem Typen ein neues Objekt erstellt, auf das dem `PluginRegistrar` zugrunde liegende Interface gecastet und zurückgegeben (siehe Codeblock 4.4). Wird ein nicht existierendes Modul angefragt, wird anstelle dessen `NULL` zurückgegeben. Es muss also immer zuerst geprüft werden, ob die angefragte Instanz `NULL` ist oder nicht. Ansonsten kann es zu Problemen bei der Programm-Ausführung und zu Instabilitäten kommen (`NULL`-Pointer Exceptions, Assertions, etc.).

Monitoring-Thread

Damit weder ein anderes System noch anderen Komponenten der Software vom Monitoring ausgebremst oder unterbrochen werden, wird dieses in einen eigenen Thread ausgelagert. Das Monitoring läuft somit parallel zu allen anderen Funktionen, kann jedoch den System-Kernel ungehindert verwenden.

```
1:  public class PluginRegistrar<T> : Object {
2:    public T? getPlugin(string name) {
3:      if (pluginExists(name))
4:        return (T)new Object(ModuleType(name));
5:      return null;
6:    }
7:  }
8:
9:  // Get an ICMP-Plugin
10: ICommModule mod = comm.getPlugin("icmp");
```

Codeblock 4.4: Nachdem alle Module geladen worden sind, kann ein Modul über die Funktion `getPlugin(string name)` angefordert werden. Existiert ein Plugin mit dem Identifier `name`, wird eine Instanz von diesem zurückgegeben. Vor der Rückgabe wird das neu erstellte Objekt auf das dem PluginRegistrar zugrundegelegte Interface gecastet.

Nach dem Start des Threads durch den System-Kernel werden alle Monitoring-Systeme und Module geladen, konfiguriert und im Speicher in einer Liste gehalten. In regelmässigen Abständen wird dann diese Liste durchlaufen und geschaut, welche Prüfung aktuell ansteht. Ob ein System zum aktuellen Zeitpunkt durch das momentane Modul geprüft werden soll oder muss, wird dabei nicht vom Monitoring-Thread bestimmt, sondern vom Modul selbst. Jedes Modul weiss, wann es zuletzt ausgeführt worden ist und wann der nächste Zeitpunkt für eine Prüfung ist (siehe Kapitel 4.1.3). Der Monitoring-Thread schaut grundsätzlich nur, ob das Modul aktuell eine Prüfung durchführt oder nicht sowie ob der Zeitpunkt (Timestamp) des Moduls abgelaufen ist. Basierend auf diesen zwei Tatsachen kann dann gesagt werden, ob eine Prüfung ansteht oder nicht.

Der Monitoring-Thread kann durch den System-Kernel über das Property `pause` unterbrochen respektive pausiert werden. Dies hat den Sinn, dass wenn zum Beispiel neu zu überwachende Hosts eingefügt werden, diese vom Monitoring-Thread zuerst geladen werden müssen (definiert über das Property `reload`). Damit es bei solchen Aktionen nicht zu Datenfehlern kommt, wird bei einem Rebuild der Überwachung dieser Thread pausiert und anschliessend wieder neu gestartet, respektive der Mainloop des Threads wird wieder ausgeführt.

Kommunikations-Thread

Damit die verschiedenen Überwachungssysteme untereinander kommunizieren können, wird der MainListener-Thread gestartet, welcher für diese Kommunikation zuständig ist. Nachrichten unter den Systemen werden mit dem verbindungslosen Protokoll UDP versendet, damit ein allenfalls nicht laufendes System nicht die komplette Überwachung

ausbremst[2]. Zudem darf ein System nicht ausgebremst werden, wenn ein anderes System ausgelastet ist und das erstellen einer Antwort mehrere Sekunden in Anspruch nimmt. Bei einer TCP Verbindung wird immer eine Antwort benötigt, was bedeutet, dass das anfragende System ausgebremst wird für die Dauer welche das ausgelastete System für die Erstellung der Antwort braucht. Bei UDP können Antworten nicht direkt sondern müssen ebenfalls als UDP-Paket versendet werden. Somit spielt es bei UDP keine Rolle, wie lange ein System für die Erstellung einer Antwort braucht.

Der Thread beinhaltet einen UDP-Listener-Socket, welcher auf alle vorhandenen NICs (Network Interface Card) gebunden ist. Das Problem bei UDP-Sockets ist, dass immer angegeben werden muss, wie viele Bytes empfangen werden sollen. Aus diesem Grund können Daten nicht einfach so wie sie vorliegen gesendet werden, sondern müssen gegebenenfalls in mehrere Pakete aufgeteilt und empfangen werden. Eine Grösse, welche sich bei UDP etabliert hat, sind 1024 Bytes. Diese Anzahl scheint Tests zufolge auf den meisten Systemen performant zu laufen, während grössere Pakete bei gewissen Betriebssystemen Schwierigkeiten bereiten[3].

Aus diesem Grund musste ein einfaches und simples Datenformat definiert werden, welches Abbildung 4.2 aufzeigt. Aktuell werden alle Daten als Klartext versendet, was nicht nur viel Datenmenge bedeutet, sondern auch sicherheitskritisch sein kann. Kommende Versionen der Software sollten nicht nur eine komprimierte oder binäre Version des Protokolls aufweisen, sondern auch eine Authentifizierung und/oder sogar eine Verschlüsselung.

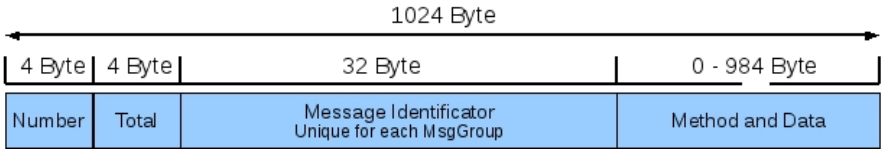

Abbildung 4.2: Jeweils 4 Zeichen definieren die aktuelle Nachrichtennummer und die totale Anzahl an Nachrichten. Die folgenden 32 Zeichen werden als Nachrichten-UID interpretiert. Die restlichen 984 Zeichen sind die Daten, welche das Kommando getrennt durch einen Doppelpunkt von den Daten beinhalten.

Alle UDP-Pakete, die auf einem definierten Port eingehen, werden empfangen und auf Gültigkeit überprüft. Ist ein Paket gültig, wird es in die einzelnen Bereiche unterteilt

[2]Bei TCP-Verbindungen kann es vorkommen, dass eine Verbindung, welche nicht korrekt oder komplett aufgebaut werden kann, diese erst nach einem definierten Timeout geschlossen wird oder so lange versucht wird diese korrekt auf zu bauen bis eben dieses Timeout abgelaufen ist. Während dieser Zeit ist das System ausgebremst, da es ja eine Antwort erwartet.

[3]Ein UDP-Paket von 2048 Bytes brauchte auf einem WindowsXP fast doppelt so lange wie zwei UDP-Pakete mit 1024 Byte.

und in einer Datenstruktur gespeichert. Diese Datenstruktur wird so lange im Speicher gehalten, bis alle Nachrichten eingetroffen sind. In einer finalen Version, werden alle empfangenen Pakete beim Absender bestätigt und in regelmässigen Abständen die vorhandenen, noch nicht kompletten Pakete, bei diesem reklamiert.

TCP-Verbindungen können einfach über die Konsole zum Beispiel mit dem `telnet`-Kommando ausgeführt und getestet werden. Dies ist bei UDP leider nicht der Fall. Eine gute Alternative bietet hierzu NetCat[net10]. Mittels diesem können UDP-Pakete auf einfache Art und Weise gesendet und empfangenen werden. Es können so andere Systeme und dessen Kommunikation simuliert werden und einfache Nachrichten wie Kommandos an ein System gesendet werden.

Durch den Befehl „`echo '1 1 74957e5872f8b604d2774b79e8c7e8c2monitor: 192.168.22.70 192.168.22.77' |nc -q 1 -u 192.168.22.27 1611`“ kann zum Beispiel unter Linux/Unix dem Überwachungs-System `192.168.22.27` mitgeteilt werden, dass es für die Systeme `192.168.22.70` und `192.168.22.77` zuständig ist. Entsprechend verhält es sich mit den anderen Kommandos.

Eine Liste mit allen Kommandos und deren Daten, ist in Tabelle 5.1 im Anhang gelistet.

4.1.2 Kommunikations-Plugins

Kommunikations-Module haben die Aufgabe, Verbindungen mit anderen Systemen her zu stellen sowie deren Antworten zu empfangen und weiter zu reichen. Dabei kann es sich sowohl um ein Netzwerk-Protokoll als auch um ein Bus-System oder eine lokale Datenanfrage handeln. Alle Kommunikations-Module werden vom Interface `ICommModule` abgeleitet (siehe Codeblock 5.5), welches wiederum eine Implementation des Interfaces `IMainModule` voraussetzt (siehe Codeblock 5.4).

Das Interface der Kommunikations-Module ist so gehalten, dass nicht bekannt sein muss, über was für eine Technologie die Daten versendet und empfangenen werden. Jegliche Einstellungen werden über die Methode `setParam(string key, string value)` definiert. Dabei kann jedes Modul selber entscheiden, welche Parameter es anschliessend verwendet und welche nicht.

Neben Funktionen zur einfachen Datenanfrage und Datenübermittlung sind auch drei Signale zur Eventbehandlung auf dem Interface enthalten. Dies ist zum einen das Signal `onError` für das Error-Handling sowie die zwei Daten-Signale `onData` und `onComplete`. Der Event `onData` wird aufgerufen, sobald Daten empfangen werden; `onComplete` erst wenn die Kommunikation beendet ist. Auf die Signale können verschiedene Methoden gebunden werden, welche beim Auftreten des Events aufgerufen werden.

Nachdem ein Modul instanziert und alle notwendigen Parameter gesetzt worden sind, kann durch die Funktion `send(string? data="")` eine Anfrage gestartet werden. Optional können beim Senden noch Daten mit angegeben werden, welche jedoch nicht von allen Modulen genutzt werden müssen.

Bei mehrstufigen Protokollen, wie zum Beispiel einer SMTP-Prüfung, kann über das `onData`-Signal eine Antwort empfangen, ausgewertet und durch einen erneuten Send-Aufruf auf die empfangenen Daten eingegangen werden. Besteht eine Anfrage nur aus einem Request, zum Beispiel ein ECHO-Ping, kann auch nur auf das `onComplete`-Signal eingegangen werden. Dieses beinhaltet alle empfangenen Daten als String-Array.

Das `onError`-Signal sollte grundsätzlich immer beachtet werden. Tritt ein Fehler auf, kann durch dieses Signal festgestellt werden, um was für einen Fehler es sich handelt und so zum Beispiel ein Alert versendet oder eine andere Methode ausprobiert werden.

Liste von verfügbaren Modulen

Die Kommunikations-Module befinden sich im Unterverzeichnis `pnetwork`:

udp Sendet Daten über einen UDP-Socket an einen Host. Benötigt werden mindestens die Parameter „host“ und „port“ sowie die Daten beim Aufruf der Send-Funktion. Aktuell wird für eine eventuelle Antwort kein Socket geöffnet. In einer kommenden Version wird dies möglich sein, um so zum Beispiel SNMP-Prüfungen direkt durchführen zu können.

tcp Sendet Daten über einen TCP-Socket und wartet eine Antwort ab. Benötigt werden mindestens die Parameter „host“ und „port“ sowie die Daten beim Aufruf der Send-Funktion. Bei der aktuellen Version bestehen noch gewisse Schwierigkeiten bei der Prüfung ob der Socket noch geöffnet ist oder nicht. Soll der Socket explizit geschlossen werden, kann dies durch einen „send()“-Aufruf (ohne Daten) forciert werden. Es kann also sein, dass momentan bei einer „falschen“ Verwendung des TCP-Modules das `onComplete`-Signal nicht aufgerufen wird.

snmp Prüft das Vorhandensein eines Systems anhand einer `sysDescr`-OID-Anfrage. Benötigt wird mindestens der Parameter „host“. Die aktuelle Version dieses Plugins funktioniert lediglich unter einem Unix-ähnlichen Betriebssystem und dem `/usr/bin/snmpget`-Befehl.

echo Sendet einen ECHO-Ping an ein System. Benötigt wird mindestens der Parameter „host“. Die aktuelle Version verwendet den `ping`-Befehl aus dem Unterverzeichnis

„bin“. Der Code des `ping`-Binaries wurde ebenfalls im Zuge der Bachelor-Thesis ausgearbeitet um eine Parser-freundlichere Ausgabe zu erhalten.

arp Sendet ein ARP-Package auf das Netzwerk und wartet die Antwort ab. Benötigt wird mindestens der Parameter „host“. Die aktuelle Version verwendet eine im Zuge der Bachelor-Thesis angepasste Version des Programms `arping2`, bei welchem alle Bindungen zu „LibNet“ entfernt wurden[4] um so von weniger externen Bibliotheken abhängig zu sein.

amods Sendet einen Befehl inklusive Daten an ein anderes Überwachungs-System. Benötigt wird mindestens der Parameter „host“ und die Daten beim Aufruf der Send-Funktion. Auf ein eventuelles Antwort-Paket wird nicht eingegangen, denn dieses wird bereits vom `MainListener`-Thread abgefangen und verarbeitet.

4.1.3 Überwachungs-Plugins

Die Überwachungsmodule sind für die Überwachung eines lokalen oder entfernten Dienstes zuständig. Diese Module sind die einzigen, welche beim Start des Systems geladen und in einer Liste gehalten werden. Dabei wird pro System und Dienst ein Modul gebraucht. Die Konfiguration der einzelnen Module wird dabei vom System ausgelesen und bei der Instanzierung des Moduls gesetzt.

Alle Überwachungsmodule müssen das Interface `IMonitorModule` implementieren (siehe Codeblock 5.6), welches wiederum das Interface `IMainModule` voraussetzt (siehe Codeblock 5.4). Mittels den Properties `running` und `next_run` muss vor einer Prüfung geschaut werden, ob diese gestartet werden soll. Durch die Methode `monitor()` wird die Prüfung anschliessend ausgeführt. Der Aufruf der Monitor-Funktion sollte asynchron (`monitor.begin()`) gestartet werden, da ansonsten die restlichen Thread-Funktionen warten bis das Modul mit der Prüfung fertig ist.

Welche Kommunikations-Module von den Überwachungsmodulen genutzt werden, liegt im Ermessen des Moduls. Diese werden bei Bedarf im System-Kernel angefragt, anschliessend konfiguriert, die Signale an Modul-Funktionen gebunden und dann Daten gesendet. Tritt bei der Verbindung ein Fehler auf, muss das Überwachungsmodul entscheiden, ob eine erneute Prüfung notwendig ist oder ob dem System-Kernel mitgeteilt werden soll, dass ein Fehler aufgetreten ist. Ob jedoch bei einem Fehler ein Alert gesendet wird oder nicht entscheidet nicht mehr das Modul, dies unterliegt der Hoheit des System-Kernels.

[4] Die benötigten Funktionen und Datenstrukturen von Libnet-1.1.5 wurden direkt in den C-Code von Arping-2.06 eingefügt.

Wird eine Überwachung erfolgreich durchgeführt und ohne Fehler abgeschlossen, wird dies dem System-Kernel durch ein Signal und einen Funktionsaufruf mitgeteilt. Dieser speichert die übermittelten Daten sowie den Zeitpunkt der Prüfung in der Datenbank.

Ist eine Prüfung abgeschlossen - ob erfolgreich oder nicht - wird das Property `running` zurückgesetzt und ein neuer Timestamp im Property `next_run` gesetzt.

Liste von verfügbaren Modulen

Die Monitoring-Module befinden sich im Unterverzeichnis `pmonitor`:

icmp Prüft das Vorhandensein eines Systems anhand eines ECHO-Pings. Erwartet als Konfiguration einen „host“ und optional ein „num“ für die Anzahl an Echo-Requests und ein optionales „timeout“, welches die Anzahl Sekunden zwischen den einzelnen Prüfungen definiert. Es wird das `echo`-Kommunikations-Plugin verwendet.

smtp Prüft einen EMail-Servers durch den Aufbau einer SMTP-Verbindung. Zur Prüfung ist ein „host“ und optional ein „port“ Parameter notwendig sowie optional ein „timeout“ und ein „num“ für die Anzahl an Verbindungsversuchen. Zusätzlich muss mit dem Parameter „mail_from“ die Absender-EMail-Adresse und mit „rcpt_to“ die Empfänger-EMail-Adresse definiert werden. Wird über „user“ und „password“ zusätzlich ein Benutzer und Passwort sowie eine Authentifizierungs-Methode mit „auth_method“ definiert, werden diese Daten ebenfalls genutzt zum Verbindungsaufbau. Die Prüfung versendet kein EMail, es wird lediglich geprüft, ob das Versenden möglich ist[5]. Es wird das `tcp`-Kommunikations-Plugin verwendet.

http Prüft ob der angegebene Webserver erreichbar ist oder nicht. Zur Prüfung ist prinzipiell nur der „host“ Parameter notwendig, optional kann noch ein „port“ definiert werden. Das Modul prüft den Server standardmässig mittels der Anfrage „OPTIONS * HTTP/1.1“ auf dessen Erreichbarkeit. Durch die Angabe des Parameters „method“, welcher die HTTP-Methode definiert (GET, POST, OPTIONS, HEAD) und einer „url“, kann aber auch die Funktionstüchtigkeit eines Webdienstes abgefragt werden. Bei der POST-Methode können zusätzlich noch Daten über den Parameter „content“ übergeben werden, welche bei allen anderen Methoden nicht beachtet werden. Als Auswertungskriterium dient in jedem Fall nur die erste Zeile

[5] Ein EMail-Server wird geprüft, indem eine Verbindung aufgebaut wird, ein Absender und Empfänger übermittelt wird sowie optional noch eine Authentifizierung. Anschliessend wird die Verbindung abgebrochen mit dem Befehl `quit`.

der Antwort, welche üblicheweise den Status als Zahl und Text beinhaltet: „HTTP/1.0 **200 OK**“ oder bei einem Fehler „HTTP/1.0 **400 Bad Request**“. Es wird das `tcp`-Kommunikations-Plugin verwendet.

4.1.4 Datenspeicherungs-Plugins

Die Module zur Datenspeicherung werden genutzt, um Daten in einer Datenbank oder auf einem entfernten Server ab zu legen und wieder aufzurufen. Das Datenmodell der einzelnen Tabellen wurde dabei absichtlich sehr einfach gehalten (Erste Normalform). Es können dadurch nicht nur Datenbanken verwendet werden, welche SQL verstehen, sondern auch andere Datenspeicherungen wie zum Beispiel einfache XML-Dateien. Die vorhandenen Tabellen und Schemata sind im Anhang aufgeführt, siehe Tabelle 5.2, 5.3, 5.4, 5.5, 5.6, 5.7 und 5.8.

Alle Datenmodule müssen das Interface `IDataModule` implementieren (siehe Codeblock 5.7), welches wiederum das Interface `IMainModule` voraussetzt (siehe Codeblock 5.4).

Im `IDataModule`-Interface sind noch weitere spezielle Konstanten und Datentypen definiert. Die Konstanten `TABLE_XX` und `SCHEMA_XX` werden benötigt um die Tabellen und deren Felder zu definieren. Die verschiedenen Tabellen werden durch die Enumeration `DataType` definiert (`CONF, MONITOR, ALERT, ...`). Jede so definierte Tabelle muss vom Datenmodul bei der Initialisierung geprüft und gegebenenfalls erstellt werden. In einem ersten Schritt muss vom Plugin, basierend auf der Schema-Definition und des Datentypen `SchemaField`[6], eine Liste mit allen Feldern erstellt werden, welche auf der Tabelle vorhanden sein müssen. Um diesen Prozess zu vereinfachen, ist auf der `SchemaField`-Struktur eine Methode vorhanden, welche einen Schema-Definitions-String parst und wiederum ein `SchemaField` zurück gibt. Die Erstellung dieser Schema-Definition wurde durch das evolutionäre Entwicklungsmodell so geboren, wird jedoch in einer kommenden Version direkt definiert werden, um das Laden resourcen-schonender zu gestalten.

Um ein Datenmodul zu verwenden, muss nach der Initialisierung definiert werden, um was für Daten sich das Plugin kümmern soll und in welcher Tabelle diese vorhanden sind. Die Definition des Typs geschieht über das Property `data_type`, welches als Wert einen `DataType` erwartet. Anschliessend kann über die `open(string db_name)`-Methode die Datenbank verbunden werden.

Um Daten zu lesen, stellt das Interface die Methoden `setWhere(string field,`

[6] Der Datentyp `SchemaField` beinhaltet alle notwendigen Daten um mittels SQL eine Tabelle erstellen zu können.

`string value)`[7] sowie `select(string? fields=„", string? groupedBy=„")` zur Verfügung. Über diese Methoden können Bedingungen sowie die Felder und Gruppierungen angegeben werden, welche aktuell benötigt werden. Anschliessend kann über die Methode `hasNext()`, `getCurrent()` sowie `getValue(string field)` auf die Felder und deren Werte zugegriffen werden (Siehe Codeblock 4.5 für ein Beispiel).

```
1:  string dbf = getDataFile("config.sqlite").get_parse_name();
2:  IDataModule conf = data.get_plugin("sqlite");
3:  if (conf != null) {
4:    conf.data_type = IDataModule.DataType.MODULES;
5:    conf.setWhere("host", host);
6:    conf.setWhere("module", module);
7:    if (conf.open(dbf) && conf.select()) {
8:      while (conf.hasNext()) {
9:        // "m" is an IMonitorModule Instance
10:       m.setParam(mod_conf.getValue("key"),
11:                  mod_conf.getValue("value"));
12:       // ...
13:     }
14:   }
15: }
```

Codeblock 4.5: Beispiel: Verwendung eines Datenmoduls zum Auslesen aller Parameter zur Konfiguration eines Überwachungsmodules. Nach dem das Modul geladen ist, wird definiert, dass Daten aus der `MODULES` Tabelle geladen werden sollen, und zwar nur jene, welche im Feld `host` und `module` entsprechende Werte beinhalten. Über die Schlaufe und die Funktion `hasNext()` werden alle Datensätze ausgelesen und die Werte „key“ und „value“ zur Parametrierung eines anderen Modules genutzt.

Um Daten über ein Datenmodul zu speichern, wird nur die Methode `save(IData Module.DataSet[] values, IDataModule.DataSet[]? where=null)` benötigt. Die Felderliste `values` beinhaltet dabei alle Felder und die dazugehörenden Werte für einen Datensatz. Wird keine Liste mit Bedingungen angegeben (`where`-Parameter), wird ein neuer Datensatz eingefügt. Andererseits werden alle Datensätze aktualisiert, welche mit der Bedingungsliste übereinstimmen. Überschriebene Daten können nicht wiederhergestellt werden. Ein Beispiel wie ein Datensatz eingefügt/aktualisiert werden kann ist in Codeblock 4.6 zu sehen.

Sollen Daten gelöscht werden, kann dies über die Funktion `delete(IDataModule. DataSet[] where)` gemacht werden. Es werden alle Datensätze entfernt, welche den angegebenen Bedingungen entsprechen. Wenn alle Datensätze einer Tabelle entfernt werden

[7] Die Methode `setWhere` wird genutzt um eine oder mehrere Bedingungen für eine Datenselektion zu definieren. Die Angaben haben keinen Einfluss auf das Löschen, Einfügen und Aktualisieren von Datensätzen.

```
1:  string dbf = getDataFile("config.sqlite").get_parse_name();
2:  IDataModule conf = data.get_plugin("sqlite");
3:  if (conf != null) {
4:    // Data-Structure with all fields and values to use for a save/
          update
5:    IDataModule.DataSet[] dataSet = new IDataModule.DataSet[] {
6:      IDataModule.DataSet() { field="module", value="icmp" },
7:      IDataModule.DataSet() { field="host", value=host },
8:      IDataModule.DataSet() { field="key", value="timeout" },
9:      IDataModule.DataSet() { field="value", value="10" }
10:   };
11:
12:   conf.data_type = IDataModule.DataType.MODULES;
13:
14:   // Insert a new DataSet
15:   conf.save(dataSet);
16:
17:   // Update an existing DataSet based on the second Data-Structure
18:   conf.save(dataSet, new IDataModule.DataSet[] {
19:     IDataModule.DataSet() { field="host", value=host },
20:     IDataModule.DataSet() { field="key", value="timeout" }
21:   });
22: }
```

Codeblock 4.6: Beispiel: Zuerst wird ein neuer Datensatz in die Modules-Tabelle eingefügt. Anschliessend werden alle Datensätze aktualisiert, welche im Feld „host“ und „key“ entsprechende Werte haben.

sollen, kann dies durch die Funktion `empty()` gemacht werden. Gelöschte Daten können nicht wiederhergestellt werden.

Liste von verfügbaren Modulen

Die Datenbank-Module befinden sich im Unterverzeichnis `pdata`:

sqlite Die Datenspeicherung erfolgt über eine Sqlite3-Datenbank im Verzeichnis „data“.

4.1.5 Alerting-Plugins

Alerting-Module werden verwendet um Meldungen über verschiedene Kanäle zu versenden. Dabei kann von den verschiedenen Kanälen nicht immer gewährleistet werden, dass eine Meldung auch wirklich versendet worden ist, oder dass dies geprüft werden kann. Daher wird grundsätzlich ein erfolgreiches Versenden als Gegeben angenommen.

Ein Alert-Modul wird definiert durch das Interface `IAlertModule` (siehe Codeblock 5.8), welches ebenfalls die Implementierung des `IMainModule` voraussetzt (siehe Code-

block 5.4). Ein Alert-Modul ist eines der einfachsten, da es lediglich dazu da ist, eine Nachricht zu versenden und anschliessend ein Signal aufzurufen. Die Prüfung, ob und wann ein Alert schon versendet worden ist, muss vom System-Kernel entschieden werden.

Das Senden eines Alerts unterliegt dabei folgendem Schema: Der System-Kernel nimmt eine Anfrage zum Versenden einer Nachricht entgegen. Anhand der Daten, dem Typ und dem Schweregrad entscheidet dieser basierend auf der Konfiguration, über welches Modul und an welche(n) Empfänger eine Nachricht versendet werden soll. Ein Alert-Modul wird erstellt und das Signal `onSent` des Moduls an eine Funktion gebunden sowie die Konfiguration mittels der Methode `setParam(string key, string value)` definiert. Anschliessend werden die Daten dem Modul zum Senden übergeben. Nach dem die Daten vom Modul versendet worden sind, wird der `onSent`-Event mit dem Empfänger, einem Status und dem Alert-UID aufgerufen. Optional kann noch eine Meldung übergeben werden.

Kann bei der verwendeten Technik nicht festgestellt werden, ob die Daten erfolgreich versendet worden sind, wird der `onSent`-Event einfach direkt nach dem Versenden aufgerufen.

Tritt beim Versenden der Nachricht ein Fehler auf, wird der `onSent`-Event ebenfalls aufgerufen. In diesem Fall jedoch mit einem Fehler-Code und einer Fehlernachricht.

Die Methode, welche an das `onSent`-Signal gebunden wird, muss anschliessend entscheiden, ob ein weiterer Alert versendet werden muss oder nicht. Dies hat ebenfalls nicht das Alert-Modul zu entscheiden.

Liste von verfügbaren Modulen

Die Alerting-Module befinden sich im Unterverzeichnis `palert`:

email Ein Alert wird mittels EMail versendet. Die Plugin-Konfiguration erfordert mindestens einen Host sowie eine oder mehrere Empfänger-Adressen. Optional kann ein Port sowie eine Absender-Adresse, Login-Daten und Login-Methoden angegeben werden. Es wird das `tcp`-Kommunikations-Plugin verwendet.

paging Ein Alert wird als Paging-Nachricht versendet. *Dieses Modul wird nicht im Rahmen der Bachelor-Thesis ausgearbeitet.*

aspsms Ein Alert wird als SMS über den SMS-Provider ASPSMS[8] versendet. *Dieses Modul wird nicht im Rahmen der Bachelor-Thesis ausgearbeitet.*

[8] ASPSMS (http://www.aspsms.ch/) ist ein Online-Dienst über welchen SMS kostengünstig versendet werden können. Die Dienstleistung wird von der in St. Gallen ansässigen Firma Vadian.Net AG unterhalten.

4.1.6 Remote-API-Plugins

Module für den entfernten Zugriff werden durch das Interface `IRemoteModule` definiert (Siehe Codeblock 5.9), welches wiederum das Interface `IMainModule` voraussetzt (siehe Codeblock 5.4). Im Unterschied zu anderen Modulen müssen die Remote-API-Plugins auf eingehende Verbindungen reagieren können. Für jedes Plugin wird, nachdem alle Module geladen sind, ein eigener Kommunikations-Thread gestartet, welcher einen Listener-Socket beinhaltet.

Die Remote-API-Plugins dürfen nicht verwechselt werden mit dem `MainListener`-Thread: Der `MainListener`-Thread dient zur Kommunikation mit anderen Überwachungs-Systemen; Die Remote-API-Module dienen zur Kommunikation mit unterschiedlichen Clients und Systemen.

Muss zwischen unterschiedlichen Anfragen/Benutzern unterschieden werden, müssen die Module Session-Fähig sein: Wird auf einem Socket eine Verbindung registriert, werden zuerst alle Daten sowie der Absender von diesen empfangen sowie ein Absender-Hash durch eine Methode auf der Thread-Instanz berechnet[9]. In einer Hash-Liste wird anschliessend nach dem Modul des errechneten Absenders gesucht. Sofern keine Instanz in der Liste vorhanden ist, wird ein neues Modul angelegt und unter dem Absender in der Liste gespeichert. Anschliessend werden dem Modul die empfangenen Daten übergeben, welche von diesem dann verarbeitet werden.

Je nach Technologie kann ein Antwort-Paket direkt über den bestehenden oder über einen neu erstellten Socket versendet werden[10]. Diese Tatsache wird durch das jeweilige Modul bestimmt, aber durch den jeweiligen Thread behandelt. Jedes Modul besitzt also die Fähigkeit, einen solchen Listener-Thread zu starten und selbst als Instanz innerhalb eines solchen Threads zu existieren.

Nachdem alle Module durch den PluginRegistrar geladen worden sind, wird je eine Instanz jedes Moduls geladen und in einer Liste im System-Kernel gespeichert. Auf diesen Instanzen wird anschliessend durch die Methode `startThread()` ein Thread gestartet, wobei das Thread-Management das jeweilige Modul übernimmt. Durch diesen Ablauf können die Module die komplette Kommunikation selber definieren, ohne dass von Aussen darauf Einfluss genommen wird. Grundsätzlich müsste das Interface `IRemoteModule`

[9] Würde ein Empfänger nur basierend auf der IP-Adresse identifiziert, könnte der Fall auftreten, dass unterschiedliche Clients aus dem gleichen durch NAT abgeschirmten Netzwerk als gleich angeschaut werden würden. Aus diesem Grund sollte der Absender-Hash nicht nur auf der IP-Adresse basierend berechnet werden

[10] Beim Verbindungs-Orientierten Protokoll TCP muss eine Antwort direkt in den bestehenden Socket geschrieben werden. Beim Verbindungslosen Protokoll UDP hingegen, kann der Listener-Socket die ganze Zeit Daten empfangen und die zu sendenden Pakete müssen jeweils über eigene Sockets versendet werden. Andere Technologien sind wiederum anders oder aber ähnlich vom Prinzip her.

keine weiteren Methoden oder Signale definieren, da die Module prinzipiell als eigenständige Programme fungieren. Lediglich die Methode `setParam(string key, string value)` wird bei der Instanzierung noch zur Konfiguration benötigt.

Liste von verfügbaren Modulen

Die Remote-API-Module befinden sich im Unterverzeichnis `premote`:

json-rpc Dieses Modul stellt einige Funktionen unter dem JSON-RPC-v2 [jso10a] Standard zur Verfügung. In erster Linie dient dieses Modul zur einfacheren Darstellung und Erfassung von Netzwerken, Systemen und Diensten sowie deren Parametern. Zusätzlich kann ein Discovery und ein Rebuild initialisiert werden. JSON [jso06] (JavaScript Object Notation) definiert einen Standard, wie primitive Javascript-Objekt serialisiert und deserialisiert werden können. JSON-RPC beschreibt den Standard des Remote-Procedure-Call basierend auf dem JSON-Schema. Eine SMD [jso10b] (Service Mapping Description) Datei beschreibt, ähnlich wie ein WSDL bei SOAP, im JSON-Format die Funktionen deren Parameter und Rückgabewerte. Das aktuelle SMD ist im Sourcecode einsehbar und wird an dieser Stelle nicht weiter beschrieben. Die Aktuelle Implementierung beinhaltet weder eine Authentifizierung noch ein Session-Handling. Es werden also keine Daten in einer Hash-Liste/Session gespeichert, wie dies im dritten Absatz bei 4.1.6 beschrieben wurde.

4.1.7 Automatisches Host-Discovery

Das automatische Auffinden von Systemen im gleichen Netzwerk kann auf zwei verschiedene Arten initiiert werden. Die erste Möglichkeit ist, dass ein entferntes System eine IP-Adresse und eine Netmask über den `MainListener` sendet. Die zweite Variante wird über verschiedene Remote-API-Module zur Verfügung stehen. Beide Methoden erstellen eine Anfrage über die zentrale Message-Queue.

Das Auffinden von laufenden Systemen in einem Netzwerk kann unter Umständen eine Menge an Netzwerk-Anfragen erfordern. Jede berechnete IP-Adresse muss separat angewählt und anschliessend geprüft werden, ob das System Antwort gibt. Damit mehrere Anfragen gleichzeitig gestartet werden können, müssen mehrere Anfragen gleichzeitig ausgeführt werden können. Das Hauptsystem darf dabei nicht behindert werden. Durch eine Aufteilung der Anfragen auf mehrere zusätzliche Threads kann dies gewährleistet werden. Jedem Thread wird eine Start-IP Adresse als Zahl sowie die Anzahl gleichzeitig laufender Prüfungen und die letzte IP-Adresse übergeben. Jeder Thread inkrementiert nach einer Prüfung die Start-IP Adresse um die Anzahl gleichzeitiger Prüfungen, um

die nächste zu prüfende IP-Adresse zu erhalten. Dies wird so lange gemacht, bis die zu prüfende IP-Adresse grösser als die letzte angegebene ist.

Wird ein laufendes System gefunden, wird dieses in einer Liste gespeichert und mit der nächsten Adresse weiter gemacht. Sind alle IP-Adresse geprüft, werden beim System-Kernel die Dienste angefragt, auf welche die laufenden System geprüft werden sollen. Dies geschieht über die Methode `getDiscoverServiceList()`, welche eine `Helper.NetService`-Liste zurück liefert (siehe Codeblock 5.2 im Anhang). Basierend auf diesen Daten werden alle gefundenen Systeme geprüft. Ist ein Dienst verfügbar, wird das auf dem Host vermerkt. Sind alle Systeme und Dienste geprüft worden, werden diese dem System-Kernel über die Methode `setDiscoveredHosts(Helper.NetService[] list)` mitgeteilt[11].

Der System-Kernel durchläuft die empfangene Liste und entscheidet, welches der gefundenen Systeme überwacht werden soll und welches nicht. In der Version, welche für die Bachelor-Thesis ausgearbeitet wird, werden alle Systeme aufgenommen und per ECHO-Ping überwacht. Zusätzlich werden alle Überwachungs-Systeme in eine weitere Tabelle geschrieben, um zu einem späteren Zeitpunkt eine automatische Überwachungspartner-Zuweisung aus zu führen.

4.1.8 Automatische Zuweisung der Überwachungspartner

Die automatische Zuweisung der Überwachungspartner erfolgt jeweils pro Netzwerk, alle Systeme eines Netzwerkes werden also zu einem Hypercube zusammengefasst. Die einzelnen Netzwerke werden anschliessend wiederum genutzt, um einen weiteren Hypercube zu bilden, welcher für die Überwachung der Netzwerkverfügbarkeit dient. Dienste, welche von extern überwacht werden müssen, werden in einer zukünftigen Version in die System-Hypercubes integriert. Die Definition und Technik wird zu einem späteren Zeitpunkt ausgearbeitet.

Die Berechnung der Anzahl Verbindungen, Anzahl Knoten und so weiter kann durch einfache Funktionen gemacht werden, wie in Kapitel 3.4.2 bereits gezeigt worden ist. Ein grösseres Problem stellt die algorithmische Erstellung eines Hypercubes dar, denn darüber schweigen sich alle gefundenen Dokumente und Informationen aus.

Zur Bildung eines Hypercube muss zuerst die Dimension und die Anzahl Nodes be-

[11] Dieser Vorgang unterscheidet sich zu dem in Abbildung 3.1 vorgeschlagenen Szenario dahingehend, dass die Prüfung der Dienste nicht nach dem Auffinden eines, sondern erst dann gemacht wird wenn alle Systeme gefunden sind. Rein technisch gesehen spielt es keine Rolle, wann welche Prüfungen gemacht werden. Während dem Ausprogrammieren hat sich dieser Weg jedoch als „logischer“ heraus kristallisiert, da zuerst die laufende Aufgabe abgeschlossen (und nicht pausiert) werden soll bevor mit der nächsten angefangen wird.

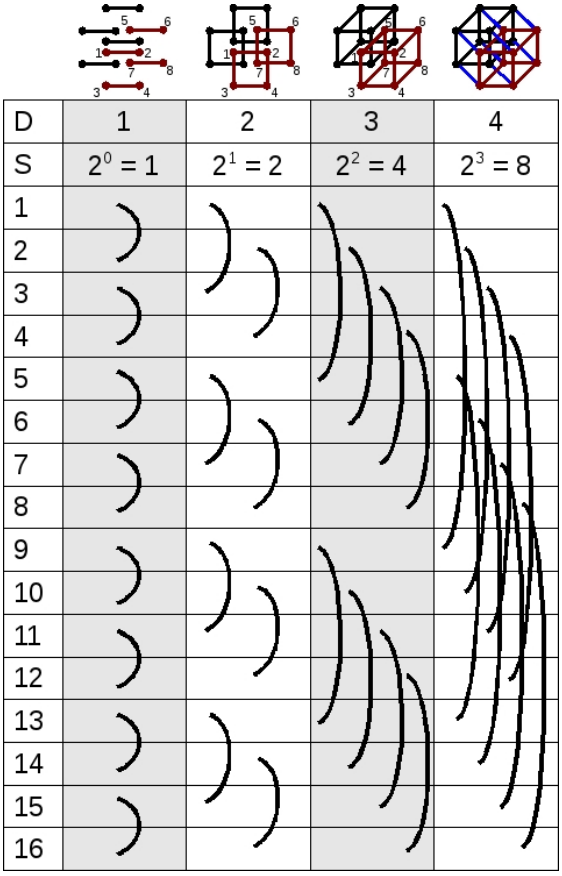

Abbildung 4.3: Darstellung des Algorithmus zur Erstellung eines Hypercube der vierten Dimension. In jeder Dimension D werden jeweils $S = 2^{D-1}$ Knoten nacheinander mit den S nächsten Verbunden. Anschliessend werden S Knoten übersprungen und die noch ausstehenden wie vorhin verbunden. Codeblock 4.7 zeigt ein Code-Beispiel dieses Vorganges.

kannt sein. Die Dimension legt dabei fest, wie viele Verbindungen jede Node aufweist, also wie viele Schritte zur Berechnung notwendig sind. Bei einem Hypercube der dritten Dimension (ein Würfel) weist jede Node drei Verbindungen auf. Es sind also drei Durchgänge notwendig. Der Prozess fängt bei der ersten Node im ersten Durchgang an. Diese erste Node wird mit der nachfolgenden verbunden, also Node zwei. Da die zweite Node nun schon eine Verbindung aufweist, muss diese übersprungen werden und mit der dritten weitergemacht werden. Diese wird anschliessend wiederum mit der nachfolgenden, also der vierten Node verbunden. Dieser Prozess wird so lange wiederholt bis alle Nodes eine Verbindung aufweisen.

Beim zweiten Durchlauf wird wieder mit der ersten Node angefangen, jedoch wird dieses mal nicht erneut eine Verbindung mit der zweiten Node hergestellt, sondern mit der Dritten. Die zweite Node besitzt noch keine zwei Verbindungen, also ist diese als

nächste dran und wird mit der vierten Node verbunden. Node drei und vier weisen nun schon je zwei Verbindungen auf, es wird also mit der fünften Node weitergemacht und diese mit der Siebten verbunden. Nach dem zweiten Durchlauf weisen alle Nodes je zwei Verbindungen auf. Auf dem Papier wären nun also zwei Quadrate zu sehen.

Diese beiden Quadrate müssen im dritten Durchlauf nun zu einem Würfel zusammengesetzt werden, was wieder nach dem gleichen Prinzip geschieht. Begonnen wird wieder mit der ersten Node. Diese muss nun in die dritte Dimension verbunden werden, also mit der fünften Node. Die zweite Node muss eine Verbindung zur sechsten, die dritte eine zur siebten und die vierte eine zur achten Node aufbauen. Um die nächste Dimension zu verbinden, was dann die 4. Dimension wäre, sind nicht mehr je vier Nodes zu verbinden und zu überspringen, wie in der dritten, sondern je acht Stück.

Darauf aufbauend kann der Algorithmus in folgenden wenigen Aussagen zusammengefasst werden:

- Im ersten Durchlauf wird jeder Knoten mit dem Nachfolgenden verbunden, wenn dieser nicht schon eine Verbindung aufweist. Es entstehen folgende Verknüpfungen: $1 \rightarrow 2$, $3 \rightarrow 4$, $5 \rightarrow 6$, $7 \rightarrow 8$, ...
- Im zweiten Durchlauf wird jeder Knoten mit dem übernächsten verbunden, wenn dieser nicht schon eine Verbindung aufweist. Es entstehen folgende Verknüpfungen: $1 \rightarrow 3$, $2 \rightarrow 4$, $5 \rightarrow 7$, $6 \rightarrow 8$ ...
- Der dritte Durchlauf ist entsprechend, nur dass jeweils 3 Knoten ausgelassen werden. Die Verknüpfungen sind entsprechend: $1 \rightarrow 5$, $2 \rightarrow 6$, $3 \rightarrow 7$, $4 \rightarrow 8$ ...
- Im vierten Durchlauf werden dann nicht mehr 3 Knoten ausgelassen, sondern total sieben.

Pro Durchlauf müssen also immer $2^{dimension-1}$ Nodes verbunden und anschliessend auch $2^{dimension-1}$ Nodes übersprungen werden. Eine visuelle Darstellung des Prozesses ist in Abbildung 4.3 zu sehen. Aus der Grafik wird auch ersichtlich, dass die Sprünge pro Dimension jeweils um eine Zweierpotenz höher sind.

Dieser Algorithmus kann in dieser Form einfach und schonend in ein Programm umgesetzt werden. Ein Beispiel zeigt Codeblock 4.7, wobei bei dieser Implementierung zusätzlich noch leere Knoten berücksichtigt werden. Es wird davon ausgegangen, dass bei der Verbindung der Knoten durch die Funktion `connectNode(node)` geschaut wird, ob der übergebene Knoten bereits eine Verbindung mit dem zu verbindenden Knoten aufweist oder nicht. Nur wenn noch keine Verbindung besteht, wird eine hergestellt (siehe

```
1:  for (dim = 0; dim < dimension; dim++) {
2:    dim_2 = Math.pow(2, dim);
3:    for (i = 0; i < num_nodes; i += (2*dim_2)) {
4:      for (j = 0; j < dim_2; j++)  {
5:        if (node_list[i+j] != null) {
6:          _n = (i + j + dim_2) % num_real_nodes;
7:          node_list[i + j].connectNode(list[_n]);
8:        }
9:      }
10:   }
11: }
```

Codeblock 4.7: Algorithmus zur Erstellung eines Hypercubes basierend auf einer Liste von Nodes. Die Prüfung auf „null“ bei der Verknüpfung behebt das Problem der leeren Knoten. Das Modulo mit der Anzahl existierender Knoten (Zeile 6) beim zu verbindenden Knoten gewährleistet, dass keine „null“-Node übergeben wird und dass jede Node mehr als nur eine Verbindung aufweist.

Codeblock 4.8). Würde dies nicht geprüft, könnte ein Knoten mehrfach mit dem gleichen Knoten verbunden werden und es müsste mittels einem zusätzlichen Parameter definiert werden, ob der aktuelle Aufruf aus dem Algorithmus oder aus der Funktion kommt. Nur wenn der Aufruf direkt vom Algorithmus gemacht würde, dürfte der übergebene Knoten mit dem aktuellen verbunden werden (Zeile 12). Ansonsten würden sich die Verbindungsfunktionen auf den zwei Knoten endlos gegenseitig aufrufen.

```
1:  if (node == null) return;
2:  if (node == this) return;
3:  bool connected = false;
4:  foreach (HyperNode c_node in node_list) {
5:    if (c_node == node) {
6:      connected = true;
7:      break;
8:    }
9:  }
10: if (!connected) {
11:   node_list.append(node);
12:   node.connectNode(this);
13: }
```

Codeblock 4.8: Die Funktion, welche zwei Knoten verbindet, prüft zuerst ob der Knoten nicht schon in der Liste vorhanden ist. Wenn nicht, wird der Knoten eingefügt und die gleiche Funktion auf dem zu verbindenden Knoten aufgerufen. Ohne diese Prüfung würde hier eine endlose Aufruf-Aktion stattfinden.

4.1.9 Versenden von Alert-Nachrichten

Das Problem der mehrfachen Meldungs-Zustellung wurde bereits in Kapitel 3.5 diskutiert und dazu eine akzeptable Lösung gefunden. Das System schaut in einem ersten Schritt in der Datenbank nach, wann dieser Alert (basierend auf Host und Modul) zuletzt gesendet worden ist. Anhand des Zeitstempels sowie des Prüfintervals wird dann entscheiden, ob die aktuelle Meldung notwendig ist oder nicht.

Die Benachrichtigung, welches Modul bei welchem System fehlgeschlagen ist, wird durch den `MainListener` empfangen und durch das `MainSystem` in die Datenbank eingetragen. Die Alert-Funktion `sendAlert` liest diese Daten aus und entscheidet, ob eine Meldung versendet werden soll. Soll ein Alert versendet werden, wird dies zuerst den anderen Überwachungssystemen mitgeteilt und anschliessend die Alert-Meldung versendet. In Abbildung 3.3 wird zuerst die Meldung versendet und anschliessend die anderen Systeme darüber informiert. Dies kann jedoch dazu führen, dass eine Meldung mehrfach versendet wird, denn während das eine System eine Nachricht sendet, können andere den Ausfall ebenfalls bemerken und eine Meldung versenden.

Grundsätzlich würde eine Meldung unter Umständen alle paar Sekunden versendet werden, denn bei der ausgearbeiteten Lösung wird nur basierend auf dem Prüfungsintervall eine Pause definiert. Um diesem Problem Herr zu werden, wird versucht heraus zu finden, wie lange das System bereits nicht mehr auf den Dienst reagiert. Basierend auf dieser Anzahl Sekunden kann anschliessend ein Alert-Plan geladen werden.

Ein Alert-Plan, definiert durch die `AlertConfig`-Klasse, beschreibt, wie mit einem Ausfall bei einem Modul und System sowie der entsprechenden Downtime umgegangen werden soll. Die Konfiguration beinhaltet den Namen des Überwachungsmodules, die Sekundenwerte „von“ und „bis“ zur Definierung der Downtime, ein Timeout in Sekunden sowie eine Liste von Empfängern. Basierend auf dem Timeout und der letzten Alert-Meldung kann somit gesagt werden, ob oder wann eine Meldung versendet werden soll.

In der für die Bachelor-Thesis ausgearbeiteten Technologie-Demonstration wird der Alert-Plan eine statische Liste von Werten sein. In einer zukünftigen Version wird diese global pro Dienst definiert und auf Systemebene noch verfeinert werden können. Es kann zukünftig somit bestimmt werden, welcher Dienst über welches Alert-Modul bei welchen Personen eine Benachrichtigung hinterlassen soll.

4.2 Einfache Installation und Konfiguration

In einer finalen Version sollte das Produkt nicht nur als binäre Version, sondern auch als Quellcode verfügbar sein. Die Voraussetzung für die Compilierung ist nicht nur ein

aktueller Vala- sondern auch ein C-Compiler. Zusätzlich müssen noch verschiedene Libraries wie die GLib, GEE und GIO vorhanden sein, sowie Datenbank-Bibliotheken für die verschiedenen Module. Sind alle Voraussetzungen gegeben, kann der Quellcode durch GNU-Make kompiliert und anschliessend verwendet werden.

Ist das Produkt in einer binären Form vorhanden, muss dieses nur in einen Ordner kopiert und kann dort direkt gestartet werden. Die Grundkonfiguration, also welche Datenbank-Engine genutzt werden soll, kann über einen Startparameter definiert werden. Andere Einstellungen müssen nicht getätigt werden. Damit das Programm beim Systemstart automatisch gestartet wird, muss nur noch ein entsprechendes Kommando in den Startprozess eingefügt werden. Dieser ist nicht nur plattformabhängig sondern auch unter Linux/BSD/Unix unterschiedlich von Distribution zu Distribution.

Um ein System in die Überwachung mit auf zu nehmen, muss dieses über eine Weboberfläche oder ein anderes Remote-API-Modul konfiguriert werden. Diese Grundkonfiguration kann dabei auf zwei unterschiedliche Arten erfolgen:

Über eine direkte Verbindung kann dem neuen System mitgeteilt werden, welche IP-Adresse es hat und in welchem Netzwerk es ist. Soll das System in ein bestehendes Überwachungsnetzwerk eingebunden werden, muss dem neuen System nur ein anderes im gleichen Netzwerk vorhandenes System angegeben werden. Ist in dem neuen Netzwerk kein anderes System vorhanden, müssen eventuelle weitere Parameter angegeben werden, um zum Beispiel eine OpenVPN-Verbindung aufbauen zu können.

Über ein anderes System kann das neue System ebenfalls eingebunden werden. Auf einem System kann über ein Remote-API-Modul ein neues System durch die Angabe von IP-Adresse und Netzwerk eingebunden werden. Diese Methode funktioniert jedoch nicht, wenn das neue System das einzige in einem neuen Netzwerk ist, denn dieses besitzt noch keine OpenVPN-Daten und Verbindungen.

Ist das neue System erst einmal im Netzwerk registriert, kann die Konfiguration an diesem vorgenommen werden. Durch einfache Prozessabläufe können verschiedene Dienste konfiguriert werden. Sind alle Parameter und Dienste eines Systems konfiguriert, kann durch eine Synchronisation das entsprechende System aktualisiert werden. Eine solche Synchronisation sollte jedoch erst nach Beendigung der Konfiguration durchgeführt werden, denn durch eine Einbindung eines neuen Systems oder Dienstes, wird eine komplette Neubildung des Hypercubes fällig.

Die grundlegenden Konfigurationen wie die globalen und lokale Datenbanken, VPN-Daten etc. werden ebenfalls mittels den Remote-API-Modulen konfiguriert. Bei der Syn-

chronisation von diesen wird aber in den meisten Fällen kein Rebuild des Hypercubes notwendig. Eine Reinitialisierung des Management-Netzwerkes oder des Systems ist möglich.

4.2.1 Mögliche Funktionen einer Konfigurationsoberfläche

Eine Konfigurationsoberfläche sollte wenn möglich die nachfolgenden Funktionen und Bereiche aufweisen:

Grundkonfiguration: Die Grundkonfiguration beinhaltet alle Basis-Parameter und Definitionen. Dies sind zum Beispiel globale Parameter der einzelnen Überwachungsmodule wie ein Timeout bei einem ECHO-Request, welche Datenbank als Standard verwendet wird, ob und welche Datenbank als globaler Datenbank-Mirror/Backbone verwendet wird, etc.

Netzwerke: Die Netzwerk-Konfiguration definiert in erster Linie das Netzwerk anhand einer IP-Adresse und einer Subnet-Maske. Zusätzlich werden noch Parameter für die Herstellung eines OpenVPN-Management-Netzwerkes sowie eine Gateway-Adresse definiert, über welche die anderen Netzwerke kommunizieren können.

Network-Discovery: Das Network- oder auch Host-Discovery dient in erster Linie dazu, neue Systeme in einem Netzwerk zu finden und diese zu untersuchen. Die so gefundenen Systeme können anschliessend über die **Host-Konfiguration** bearbeitet werden.

Host-Konfiguration: Dieser Bereich listet alle Systeme und deren überwachten Dienste auf. Auf den einzelnen Systemen können neue Dienste hinzugefügt, entfernt oder bearbeitet werden. Die globale System-Konfiguration wie IP-Adresse, Netzwerk, usw. sollte ebenfalls darüber konfigurierbar sein.

4.2.2 Installation der TechDemo

Die Technologie-Demonstration soll die Machbarkeit aufzeigen und nicht ein voll funktionstüchtiges Produkt darstellen. Zur Prüfung der Funktionalität werden relativ viele Informationen auf der Konsole ausgegeben.

Die Technologie-Demonstration erfordert aktuell mehr Konfigurationsaufwand als die geplante finale Version. Dazu notwendig ist ein Sqlite3-Datenbank-Programm wie SQLiteMan (`http://www.sqliteman.com/`) um verschiedene Systeme in die Überwachung auf zu nehmen. Die TechDemo wird in zwei Binär-Versionen angeboten (32-Bit POSIX,

64-Bit POSIX) sowie auch als Vala-Sourcecode. Eine Windows-Version ist aktuell nicht vorhanden.

Die binären Versionen können aus einem beliebigen Verzeichnis gestartet werden. Als Startparameter werden die IP-Adresse als Dot-String sowie die Subnet-Maske als numerischer Wert benötigt: `./modules 1.2.3.4 24`

Vor dem ersten Start sollte die Datei „dummy_alert_config.txt“ im data-Ordner bearbeitet werden. Diese beinhaltet momentan die Konfiguration für das EMail-Alert-Modul. Die Parameter sollten selbstbeschreibend sein, eine Auflistung und Beschreibung ist aber im Anhang in Tabelle 5.9 zu sehen. Anschliessend muss die Sqlite-Datenbank „config.sqlite“, ebenfalls im data-Verzeichnis, bearbeitet werden. In der Tabelle „networks“ muss in einem ersten Schritt das bestehende Netzwerk angepasst und allenfalls zusätzliche eingetragen werden. In der Tabelle „hosts“ müssen alle anderen Überwachungssysteme eingetragen werden. Die Tabelle „modules“ schlussendlich beinhaltet alle Systeme, welche überwacht werden sollen. Jede Zeile in der „modules“ Tabelle steht für einen Parameter. Es werden somit zwei Zeilen benötigt, wenn ein System mittels `icmp` alle 5 Sekunden (timeout) und je 2 Echo-Requests (num) überwacht werden soll.

Sind alle Konfigurationen gemacht, kann das System, wie oben angegeben, gestartet werden. Kommandos können anschliessend mittels „netcat“ (siehe Kapitel 4.1.1) dem System übergeben werden. Mit der Erstellung von Test-Hosts sollte vorsichtig umgegangen werden. Da Sqlite nicht gerade die schnellste Datenbank ist, kann zum Beispiel die Erstellung eines /16-er Netzwerkes mit mehreren tausend Systemen einige Minuten in Anspruch nehmen.

Alternativ kann auch mittels einem aktuellen Firefox oder einem auf KHTML/WebKit basierenden Browser wie Chrome oder Safari eine Konfiguration über das JSON-RPC-Modul vorgenommen oder geändert werden. Hierfür muss die die Datei „json-rpc.html“ aus dem „demo“-Verzeichnis geöffnet werden. Der erste Dialog dient zur Angabe eines laufenden Systems, wobei als Standard-Port für JSON-RPC momentan der Port 2784 (www-dev) verwendet wird. Die Oberfläche dieser Demo-Weboberfläche sollte weitgehenst selbsterklärend sein: Im linken Bereich werden alle Netzwerke und durch einen Klick auf ein solches auch die Systeme angezeigt. Durch einen klick auf ein System werden mittig die System-Konfigurationen dargestellt und rechts ein Kommunikations-Log. Oberhalb wird eine Menubar/Symbolleiste angezeigt, über welche verschiedene Aktionen ausgeführt werden können. Wichtig bei der Benützung zu wissen ist, dass der Server teilweise keine Verbindungen annimmt, was zu Fehlern führen kann. Wird ein solcher Fehler gezeigt muss unter umständen die Seite neu geladen werden, da die Browser erst anschliessend wieder in der Lage sind eine Verbindung herzustellen.

Ein Problem bei der aktuellen Version stellt ein Speicherfehler in der SQLite3-Bibliothek dar. Bei der Verwendung von `Sqlite3.Statement` beim auslesen von Daten, werden Speicherbereiche nicht mehr korrekt fereigegeben. Dies hat zur Folge dass durch den Betrieb der Überwachungs-Software aktuell immer mehr Speicher (heap) verbraucht wird und nicht wiederverwendet werden kann. Durch die Verwendung von Sqlite3 als externe Bibliothek, kann es durchaus sein, dass dieser Fehler nicht auf jeder Installation vorhanden ist.

4.3 Verschiedene Szenarien

Zur Veranschaulichung der Technologie-Demonstration werden in diesem Kapitel einige Szenarien nachgespielt sowie die Reaktion des Systems darauf gezeigt.

4.3.1 Rebuild eines Hypercubes

Damit der Prozess der Bildung eines Hypercube besser gezeigt werden kann, wird eine Datenbank mit 20 Pseudo-Systemen aus dem Subnet „192.168.22.0/24“ verwendet. Diese Dummy-Hosts werden durch den Befehl `echo '1 1 a8812636a6ad0b84ab8f6943728551a4 testdata: 192.168.22.0 24 20' |nc -q1 -u 192.168.22.27 1611` angelegt und durch `echo '1 1 a8812636a6ad0b84ab8f6943728551a4rebuild:' |nc -q1 -u 192.168.22.27 1611` neu gebildet.

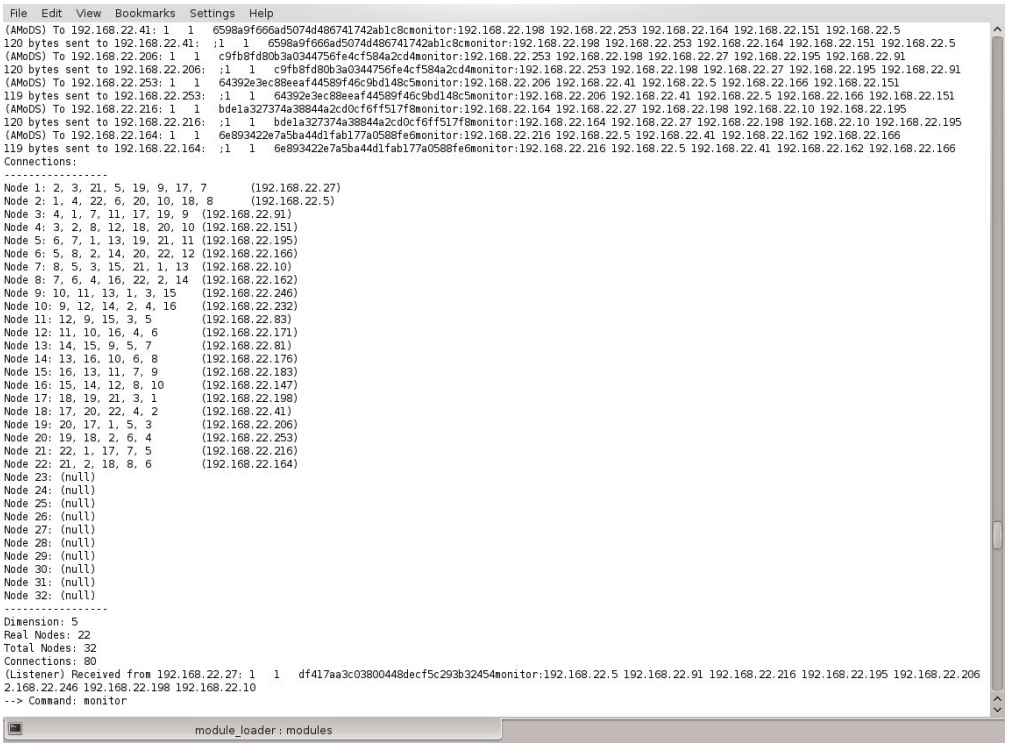

Abbildung 4.4: Im oberen Bereich sind Nachrichten sichtbar, welche die anderen Systeme darüber informieren, welche anderen Systeme sie überwachen müssen. Der mittlere Bereich zeigt den Hypercube, respektive alle gemachten Verbindungen zwischen den Systemen. Unten angefügt werden noch die Hypercube-Daten gezeigt. Die letzten Zeilen zeigen die eingegangene Nachricht, welche dem aktuellen System mitteilt, welche anderen Systeme es überwachen muss.

Achtung: Wenn Systeme manuell in die Datenbank eingetragen worden sind, die nicht im gleichen Netzwerk wie der aktuelle Server sind, werden diese aus der Überwachung entfernt. Es wird nicht das System an sich entfernt, sondern nur die Einträge in der „modules“-Tabelle, welche alle überwachten Systeme beinhaltet.

4.3.2 Ausfall eines Systems

Fällt ein System aus und wird ein Alert versendet, wird dies allen anderen Systemen mitgeteilt. Zusätzlich ist in diesem Szenario noch ein Konfigurationsfehler, welcher das senden einer Alert-Meldung nicht zulässt. Dieser Fall wird in der TechDemo jedoch nicht abgefangen, denn es wird grundsätzlich davon ausgegangen, dass ein Alert versendet werden kann.

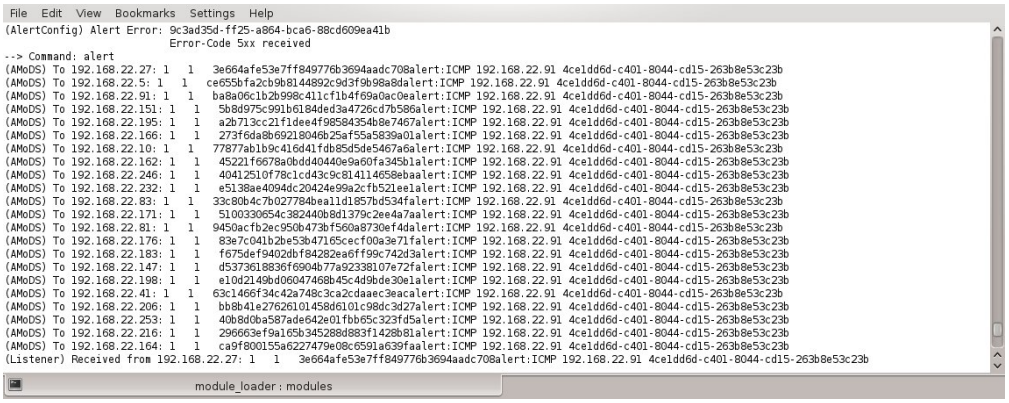

Abbildung 4.5: Die ersten Zeilen zeigen auf, dass der Alert nicht versendet werden konnte. Der EMail-Server hat einen 5xx-Code als Antwort gesendet. Die anschliessenden Zeilen zeigen die Nachrichten an alle weiteren Systeme, welche informiert werden, dass der Alert mit der ID **4ce1dd6d-c4...** für das System **192.168.22.91** und Plugin **ICMP** versendet worden ist. Die letzte Zeile zeigt auf, dass das laufende System die oben gesendete Nachricht auch empfängt und weiterverarbeitet.

Wird ein Systemfehler entdeckt, ein Alert wurde jedoch schon gesendet, wird so lange gewartet, bis wieder ein Alert versendet werden darf.

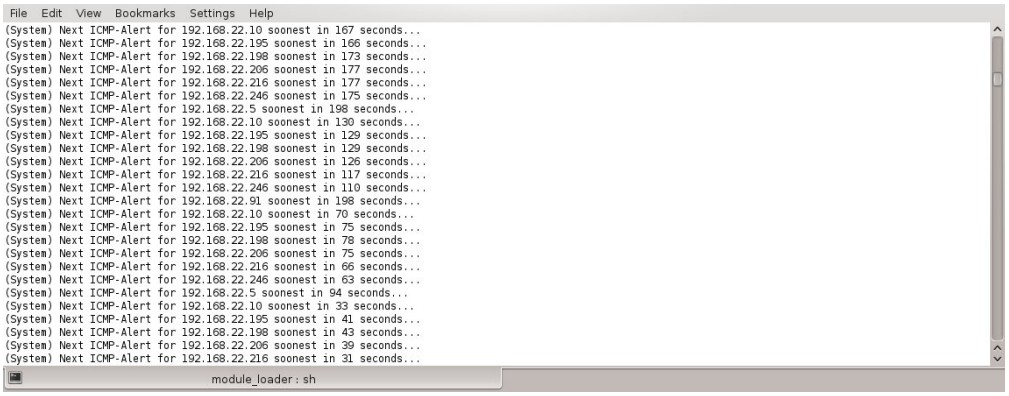

Abbildung 4.6: Wird das Senden eines Alerts beim auffinden eines Problems durch eine Semaphore geblockt, wird dies aktuell durch die Wartezeit angezeigt.

4.3.3 Überwachung eines EMail-Servers

Wird bei der SMTP-Modul-Konfiguration ein Fehler gemacht, wird dies aktuell nicht erkannt, sondern es wird angenommen, dass etwas mit dem System nicht in Ordnung ist.

Abbildung 4.7: Auf den ersten Zeilen wird gezeigt, mit welchen Parametern das SMTP-Modul geladen wird. Das fehlerhaft Passwort verursacht eine Alert-Meldung, welche anschliessend auf die umliegenden Systeme verteilt wird.

5 Fazit

Es existieren viele gute und zuverlässige Softwarelösungen zur Überwachung von Systemen und Netzwerken auf dem Markt. Die meisten, wenn nicht sogar alle, bauen auf dem Prinzip der zentralen Datenerfassung auf. Alle Systeme werden von einem zentralen Punkt aus überwacht. Nachrichten und Meldungen werden von dieser zentralen Instanz gespeichert und versendet. Einige Systeme arbeiten mit zusätzlichen Satelliten oder Agents, welche die Aufgabe haben, gewisse Netzwerksegmente zu überwachen. Diese dezentralen Instanzen sind jedoch nichts anderes als Proxys, also Schnittstellen, welche Dienste prüfen können und die Resultate an die zentrale Einheit weiterleiten.

Die Idee einer hochverfügbaren und autonomen Überwachung müsste, basierend auf diesen Fakten, eigentlich schon zu einem früheren Zeitpunkt aufgekommen sein. Bei jedem Server-Ausfall, Netzwerkzusammenbruch oder sonstigem Problem mit der zentralen Instanz wird das komplette Netzwerk nicht überwacht. Bei den Recherchen zu dieser Arbeit wurden weder Dokumente noch Abhandlungen oder Anderes gefunden, welche auf diese Idee hingewiesen oder dies sogar als Thema hatten.

Das Prinzip ist relativ einfach und schnell erklärt. Bei der theoretischen Betrachtung und der praktischen Umsetzung wurden jedoch vier kleinere Knackpunkte erkannt:

- Die Überwachung muss Netzwerk-übergreifend sein, ohne übermässige Router- und Firewall-Konfiguration.
- Wie werden die Systeme innerhalb eines Netzwerkes und wie die Netzwerke selbst automatisch verbunden und verknüpft?
- Ein Alert darf nur einmal und nicht von allen Systemen versendet werden.
- Es muss eine Möglichkeit geben, die Daten trotzdem zentral speichern zu können.

Durch einen Hinweis von Herr Frank Moehle, dass die zu überwachenden Systeme eventuell nach dem Prinzip eines Hypercube miteinander verhängt werden könnten, stellte sich schnell heraus, dass die ersten zwei Probleme in der Theorie relativ einfach zu lösen sind: Die Systeme eines jeden Netzwerkes bilden einen Hypercube. Die Netzwerke bilden ebenfalls einen. Dadurch behält man nicht nur die Anzahl an Verbindungen im

Griff, sondern die Systeme können auch automatisch fragmentiert und die Verbindungen der verschiedenen Netzwerke einfach konfiguriert werden.

Die Tatsache, dass viele Server und auch Netzwerkkomponenten heutzutage über eine Management-NIC verfügen, vereinfacht das erste Problem weiter. Durch die Kommunikation über dieses Netzwerk-Interface können Router und Firewalls genereller konfiguriert werden. Da jedoch nicht alle Rechner und Komponenten über zwei Netzwerkkarten verfügen, muss die Möglichkeit bestehen, dass die Kommunikation dennoch über einen separaten Kanal erfolgen kann. Dies kann durch die optionale Vernetzung der Systeme durch OpenVPN erfolgen.

Die einzige Schwierigkeit ergab sich anschliessend in der praktischen Umsetzung bei der Herleitung des Algorithmus zur automatischen Bildung eines Hypercubes. Darüber wurden leider weder Informationen noch Hinweise noch Anregungen gefunden. Dies bedeutete, dass dieser Algorithmus selber hergeleitet, vereinfacht und vor allem speicherschonend ausgearbeitet werden musste. Dies war eine der interessantesten Aufgaben der gesamten Thesis.

Die Problematik zur Verhinderung des mehrfachen Sendens von Alert-Nachrichten konnte durch relativ einfache Prüfungen und Notifikationen behoben werden.

Die zentrale Datenspeicherung aller Prüfergebnisse und Nachrichten konnte ebenfalls einfach gelöst werden. Alle Daten, die zentral gespeichert werden sollen, müssen einfach durch alle konfigurierten Daten-Module gespeichert werden. Dabei spielt es für die Software keine Rolle, wo und wann welche Daten gespeichert werden. Während das eine Modul alle Daten in einer lokalen Sqlite3-Datenbank speichert, schickt ein zweites Modul alle Daten zusätzlich noch in einen MySQL-Cluster oder über eine definierte Schnittstelle an ein beliebiges anderes System.

Die Frage, warum bislang noch kein solches System existiert, kann nach der Bachelor-Thesis immer noch nicht beantwortet werden. Die Probleme, welche sich bei der Planung und der praktischen Ausarbeitung ergeben haben, unterscheiden sich nicht von denen bei anderen Software-Projekten. Die Vorteile, die ein solches System bietet, übertreffen meiner Meinung nach die Vorteile zentralisierter Systeme. Dabei ist nicht nur die Hochverfügbarkeit ein Pluspunkt sondern auch die verteilte Netzwerklast, welche bei einem zentralisierten System ebenfalls zentralisiert ist.

Abkürzungsverzeichnis

AMoDS	Autonomous Monitoring of Distributed Services
API	Application Programming Interface
ARP	Address Resolution Protocol
BMC	Baseboard Management Controller
CDP	Cisco Discovery Protocol
DME	Digitale Melde Empfänger
DPI	Deep Packet Inspection
FIFO	First in first out
FME	Funk-Melde Empfänger
FTP	File Transfer Protocol
GSM	Global System for Mobile Communications
HTTP	Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol
IIS	Intrnet Information Server, Microsoft Webserver
IMAP	Internet Message Access Protocol
IP	Internet Protocol
IPMI	Intelligent Platform Management Interface
IPS	Intrusion Prevention System
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISP	Internet Service Provider
LLDP	Link Layer Discovery Protocol
MAC	Media Access Control, Hardwareadresse (Dt)
MIB	Management Information Base
MID	Man in the Middle
MMS	Multimedia Message Service
MO	Managed Object
MX	Mail Exchange
NAT	Network Address Translation

NIC ..... Network Interface Card
NRPE ..... Nagios Remote Plugin Executor
OID ..... Object Identifier
PDU ..... Protocol Data Unit
POCSAG ..... Post Office Code Standard Advisory Group
POP3 ..... Post Office Protocol v.3
RBL ..... Realtime Blacklist
RRD ..... Round Robin Database
SMBus ..... System Management Bus
SMS ..... Short Message Service
SMTP ..... Simple Mail Transfer Protocol
SMTP ..... Simple Mail Transport Protocol
SNMP ..... Simple Network Management Protocol
SQL ..... Structured Query Language
SSL ..... Secure Socket Layer
SSL-VPN ..... Secure Socket Layer Virtual Private Network
TCP ..... Transmission Control Protocol
TETRA ..... Terrestrial Trunked Radio
UDP ..... User Datagram Prodocol
VPN ..... Virtual Private Network
XSS ..... Cross-Site Scripting

Abbildungsverzeichnis

Tabellenverzeichnis

Codeblock-Verzeichnis

Literaturverzeichnis

[BLFF96] Berners-Lee, T. ; Fielding, R. ; Frystyk, H. *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945 (Informational). Mai 1996

[BW02] Blumenthal, U. ; Wijnen, B.: *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)*. RFC 3414 (Standard). Dezember 2002 (Request for Comments). – Updated by RFC 5590

[cac10] *Cacti, the complete rrdtool-based graphing solution*. http://www.cacti.net/. November 2010

[CDS74] Cerf, V. ; Dalal, Y. ; Sunshine, C. *Specification of Internet Transmission Control Program*. RFC 675. Dezember 1974

[CHPW02] Case, J. ; Harrington, D. ; Presuhn, R. ; Wijnen, B.: *Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)*. RFC 3412 (Standard). Dezember 2002 (Request for Comments). – Updated by RFC 5590

[CMPS02] Case, J. ; Mundy, R. ; Partain, D. ; Stewart, B. *Introduction and Applicability Statements for Internet-Standard Management Framework*. RFC 3410 (Informational). Dezember 2002

[Cri03] Crispin, M.: *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*. RFC 3501 (Proposed Standard). März 2003 (Request for Comments). – Updated by RFCs 4466, 4469, 4551, 5032, 5182, 5738

[CSF+08] Cooper, D. ; Santesson, S. ; Farrell, S. ; Boeyen, S. ; Housley, R. ; Polk, W. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280 (Proposed Standard). Mai 2008

[Dzu10] Dzubin, Thomas. *FPing ist ein ähnliches Programm wie Ping, jedoch können mehrere Hosts und ECHO-Requests gleichzeitig abgesetzt werden. FPing ist ausgelegt für den Einsatz in Scrips, die Ausgabe ist sehr leicht zu parsen.* http://www.fping.com/. November 2010

[FF05] Fenner, B. ; Flick, J. *Management Information Base for the User Datagram Protocol (UDP).* RFC 4113 (Proposed Standard). Juni 2005

[FGM+99] Fielding, R. ; Gettys, J. ; Mogul, J. ; Frystyk, H. ; Masinter, L. ; Leach, P. ; Berners-Lee, T.: *Hypertext Transfer Protocol – HTTP/1.1.* RFC 2616 (Draft Standard). Juni 1999 (Request for Comments). – Updated by RFCs 2817, 5785

[gee10] *Libgee ist Collection-Library basierend auf GObject der GLib. Libgee bietet HashMaps, Iteratoren, Queues und weiteres an. Diese Library ist in Vala geschrieben und kann wie jede GObject-basierende C-Library gebraucht werden.*
http://live.gnome.org/Libgee. November 2010

[gio10] *GIO ist eine auf der GLib basierende Library, welche platformübergreifende Funktionalitäten im I/O-Bereich (z.B. Datei und Netzwerkzugriffe) gewährleistet.*
http://library.gnome.org/devel/gio/stable/. November 2010

[gli10] *GLib ist eine C-Utility-Library welche Datentypen, Makros, Stringfunktionen und vieles mehr bietet. Die GLib steht unter der GNU LGPL und ist für fast jede Platform (Unix-Like, Win32, OS/2, BeOS) erhältlich.*
http://library.gnome.org/devel/glib/stable/. November 2010

[HPW02] Harrington, D. ; Presuhn, R. ; Wijnen, B.: *An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks.* RFC 3411 (Standard). Dezember 2002 (Request for Comments). – Updated by RFCs 5343, 5590

[IPM07] *IPMItool ist ein Konsolen-Programm zur Kommunikation mit einem IPMI-System.*
http://ipmitool.sourceforge.net/. April 2007

[jso06] *JSON (Javascript Object Notation) ist ein einfaches Datenaustauschformat, welches speziell für die Kommunikation zwischen einem Webbrowser und ei-*

ner Serverapplikation entwickelt worden ist. Es basiert auf JavaScript/ECMAScript und kann durch dieses einfach erstellt und ausgewertet werden. JSON-RPCWorkingGroup, http://www.ietf.org/rfc/rfc4627.txt. 2006

[jso10a] *JSON-RPCv2 definiert den RPC (Remote Procedure Call) Standard, basierend auf JSON (Javascript Object Notation).* http://groups.google.com/group/json-rpc/web/json-rpc-2-0. 2010

[jso10b] *Das JSON Schema, oder SMD (Service Mapping Description), ist ein Entwurf/Draft für die Struktur von Daten im JSON-Format. Bei SMD sollen unter anderem Methoden, Argumente und Rückgabewerte beschrieben werden, welche bei einem JSON-RPC Dienst genutzt werden können. Zukünftige Entwürfe oder finale Versionen sind bei der IETF vermerkt, dieser dritte Entwurf läuft am 26. Mai 2011 aus.* http://tools.ietf.org/html/draft-zyp-json-schema-03, http://groups.google.com/group/json-schema/web/service-mapping-description-proposal. 2010

[Kle08] Klensin, J. *Simple Mail Transfer Protocol.* RFC 5321 (Draft Standard). Oktober 2008

[LMS02] Levi, D. ; Meyer, P. ; Stewart, B. *Simple Network Management Protocol (SNMP) Applications.* RFC 3413 (Standard). Dezember 2002

[lua10] *LUA, Portgiesisch für Mond, ist eine Erweiterung für Programmiersprachen zur einfachen Implementierung von zusätzlichen Funktionen und Abläufen.* http://www.lua.org/. November 2010

[MR96] Myers, J. ; Rose, M.: *Post Office Protocol - Version 3.* RFC 1939 (Standard). Mai 1996 (Request for Comments). – Updated by RFCs 1957, 2449

[nag10] *Professionelles IT-Management System zur identifikation von Problemnen und deren Lösung.* http://www.nagios.com/products/nagioscore. November 2010

[net10] *Netcat ist ein einfaches Werkzeug um Daten auf das Netzwerk zu senden oder von diesem zu empfangen. Es wird auch als das Schweizer Taschenmesser für Netzwerke betitelt. GNU-Netcat ist eine Reimplementation der originalen Version, welche noch unter*

http://nc110.sourceforge.net/ zu finden ist.
http://netcat.sourceforge.net/. November 2010

[Oet10a] Oetiker, Tobias. *Tobi Oetiker's RRDtool.*
http://oss.oetiker.ch/rrdtool/. November 2010

[Oet10b] Oetiker, Tobias. *Tobi Oetiker's SmokePing.*
http://oss.oetiker.ch/smokeping/. November 2010

[Ope10a] *OpenIPMI sind verschiedene Linux-Kernel Patches, welche ein System komplett IPMI fähig machen. Diverse Server-Distributionen haben diese Erweiterungen bereits enthalten.*
http://openipmi.sourceforge.net/. November 2010

[ope10b] *OpenVPN ist eine Full-Featured open source SSL-VPN Lösung, lauffähig unter allen gängigen Betriebssystemen.*
http://www.openvpn.net/. November 2010

[Pos80] Postel, J. *User Datagram Protocol.* RFC 768 (Standard). August 1980

[Pos81] Postel, J.: *Internet Control Message Protocol.* RFC 792 (Standard). September 1981 (Request for Comments). – Updated by RFCs 950, 4884

[PR85] Postel, J. ; Reynolds, J.: *File Transfer Protocol.* RFC 959 (Standard). Oktober 1985 (Request for Comments). – Updated by RFCs 2228, 2640, 2773, 3659, 5797

[Pre02a] Presuhn, R. *Management Information Base (MIB) for the Simple Network Management Protocol (SNMP).* RFC 3418 (Standard). Dezember 2002

[Pre02b] Presuhn, R.: *Transport Mappings for the Simple Network Management Protocol (SNMP).* RFC 3417 (Standard). Dezember 2002 (Request for Comments). – Updated by RFCs 4789, 5590

[Pre02c] Presuhn, R. *Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP).* RFC 3416 (Standard). Dezember 2002

[Rag05] Raghunarayan, R. *Management Information Base for the Transmission Control Protocol (TCP).* RFC 4022 (Proposed Standard). März 2005

[Ric10] Rickli, Remo. *Automatische Netzwerk-Visualisierung.*
http://www.nedi.ch/. November 2010

[RM90] Rose, M.T. ; McCloghrie, K. *Structure and identification of management information for TCP/IP-based internets*. RFC 1155 (Standard). Mai 1990

[Rou06] Routhier, S. *Management Information Base for the Internet Protocol (IP)*. RFC 4293 (Proposed Standard). April 2006

[Tan03] Tannenbaum, Andrew S.: *Computernetzwerke*. 4., überarbeitete Auflage. Pearson Studium, 2003. – ISBN 978–3–8273–7046–4

[val10] *VALA ist eine C# ähnliche Programmiersprache und Compiler, welche den Sourcecode zuerst nach C wandelt und diesen C-Code anschliessend kompiliert.*
http://live.gnome.org/Vala. November 2010

[Wal19] Walser, Hans: *Der n-demensionale Hyperwürfel, 22. Basler Kolloquium für Mathematiklehrkräfte*
http://www.math.unibas.ch/~walser/Vortraege/Vortrag39/Skript/Hyperwuerfel.pdf. 2003-11-19

[wha10] *Allumfassende Netzwerk- und Systemüberwachung für Windows-Server.*
http://www.whatsupgold.com/. November 2010

[WPM02] Wijnen, B. ; Presuhn, R. ; McCloghrie, K. *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)*. RFC 3415 (Standard). Dezember 2002

[zab10] *Agent-Basiertes Opensource Enterprise Monitoring System inklusive Auto-Discovery.*
http://www.zabbix.com/. November 2010

[zen10] *Opensource Monitoring und System-Management Software.*
http://community.zenoss.org/. November 2010

Anhang

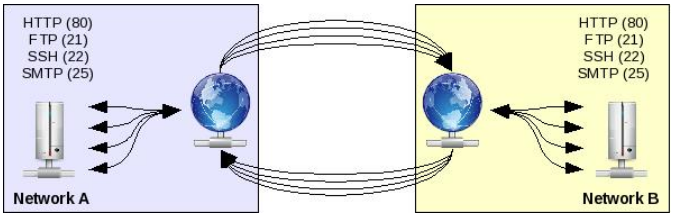

Abbildung 5.1: Beispiel für die Anzahl NAT-Regeln bei zwei Netzwerken mit jeweils einem Host, welche je vier Dienste überwacht haben wollen. Beide NAT-Firewalls müssen je vier Regeln definiert haben, um die Anfragen durch zu leiten.

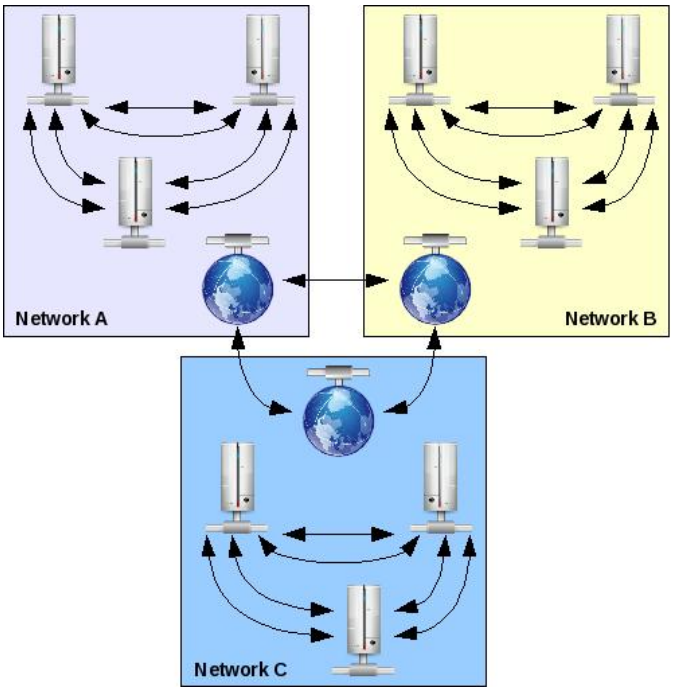

Abbildung 5.2: Beispiel für die Anzahl NAT-Regeln bei einer verteilten Überwachung. Die NAT-Boxen müssen jeweils nur eine Regel pro entferntem Netzwerk definiert haben, da dieses als Ganzes überwacht wird und nicht jedes System einzeln.

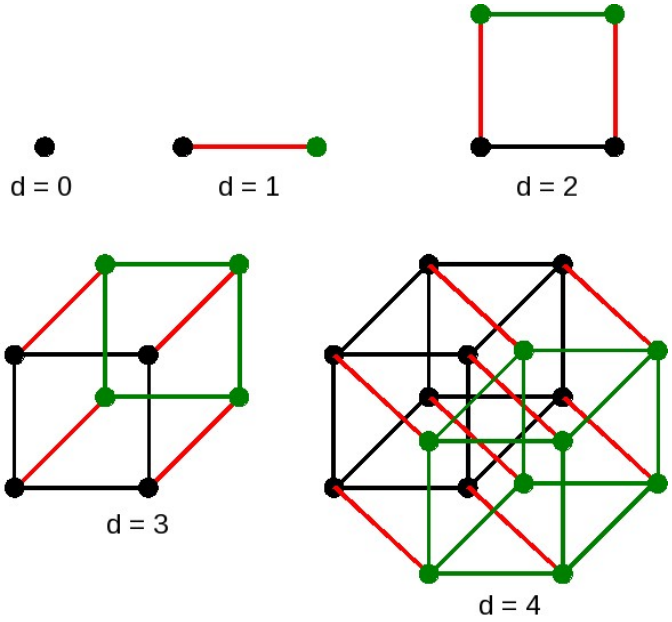

Abbildung 5.3: Hypercubes von Dimension $d = 0$ bis $d = 4$. Die Erstellung geschieht jeweils durch eine Verschiebung (grün) des vorherigen Hypercubes in eine neue Dimension sowie dem Verbinden (rot) der jeweils gleichen Eckpunkte miteinander.

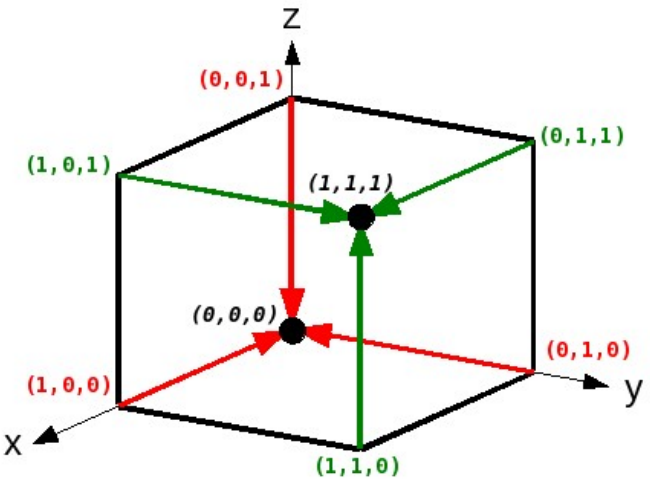

Abbildung 5.4: Die Nummerierung mit der Hamming-Distanz geschieht bei einem Würfel durch die Zuweisung der Werte $(0, 0, 0)$ und $(1, 1, 1)$ auf zwei gegenüberliegenden Ecken und anschliessend dem Verschieben der 1 resp. der 0 im Uhrzeigersinn auf den umliegenden Ecken.

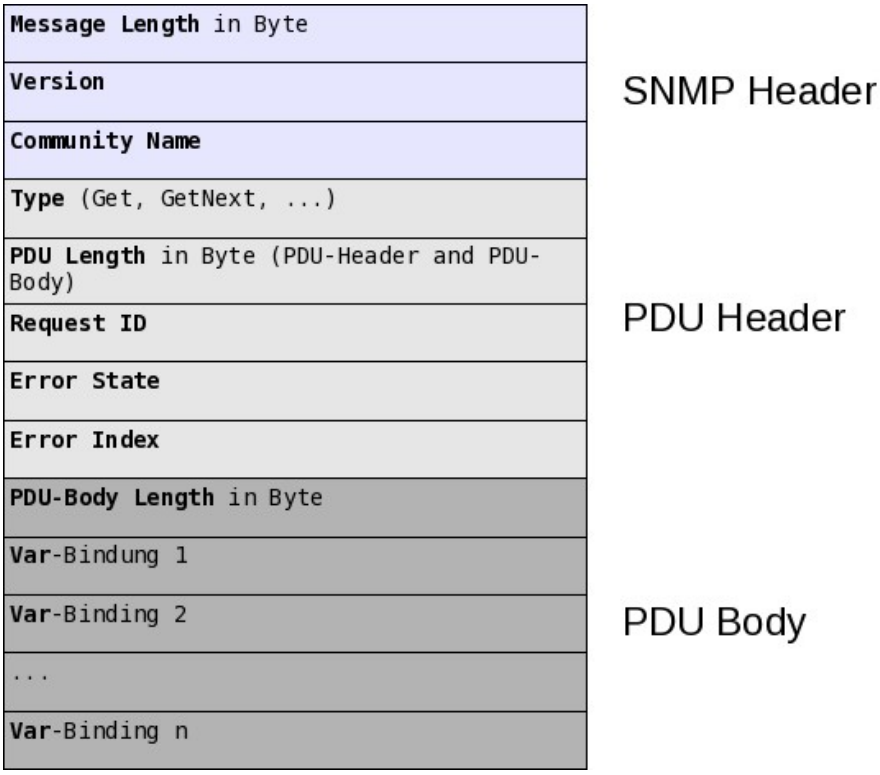

Abbildung 5.5: Aufbau eines SNMP-Pakets. Die in der MIB definierten Daten werden als MO, also Key-Value Paar, im PDU-Body nacheinander angegeben. Achtung: Der PDU-Header bei einem TRAP-Paket weist andere Daten auf, der PDU-Body ist jedoch gleich aufgebaut.

```
1: InetAddressIPv4 ::= TEXTUAL-CONVENTION
2:     DISPLAY-HINT "1d.1d.1d.1d"
3:     STATUS       current
4:     DESCRIPTION
5:         "Denotes a generic Internet address...."
6:     SYNTAX       OCTET STRING (SIZE (4))
```

Codeblock 5.1: MIB-Definition für eine IP-Adresse - definiert in der INET-ADDRESS-MIB

```
1:  public struct NetService {
2:    public string host;
3:    public string monitor;
4:    public string network;
5:    public bool available;
6:    public string to_string() {
7:      return @"host: $host\nmonitor: $monitor\nnetwork:"+
8:              "$network\navailable: %s".printf(available?"Yes":"No");
9:    }
10: }
```

Codeblock 5.2: Die Datenstruktur Helper.NetService wird verwendet um die Verfügbarkeit eines Dienstes, respektive eines Monitoring-Modules, auf einem System zu definieren.

```
1:  AsyncQueue<RequestData> request_queue;
2:
3:  class RequestData {
4:    public string command {get; private set;}
5:    public string data {get; private set;}
6:    public RequestData(string cmd, string? args="") {
7:      command = cmd;
8:      data = args;
9:    }
10: }
```

Codeblock 5.3: Die asynchrone Queue, typisiert auf `RequestData`, beinhaltet alle Nachrichten, welche nacheinander abgearbeitet werden. Die Nachrichten werden durch die Klasse `RequestData` definiert, wobei das Kommando immer angegeben werden muss, die Daten sind jedoch optional.

```
1: interface IMainModule : Object {
2:   public abstract void setRegistrar(PluginRegistrar registrar);
3:   public abstract string identifier { public get; public set; }
4:   public abstract string path { public get; public set; }
5: }
```

Codeblock 5.4: Das Haupt-Interface von Plugins beinhaltet nur Methoden zur Registrierung des Modul-Containers sowie Properties für den Identifier und den Pfad, welche in der Einstiegsfunktion des Plugins schon definiert sind.

```
interface ICommModule : IMainModule {
  public signal void onError(string ip, string data, int code);
  public signal void onData(string ip, string data, bool more);
  public signal void onComplete(string ip, string[] data,
                                int code);

  public abstract async void send(string? data="");
  public abstract void setParam(string key, string value);
  public abstract string getParam(string key);
}
```

Codeblock 5.5: Das Interface eines Netzwerk- und BUS-Modules. Die Signale `onError`, `onData` und `onComplete` können auf mehrere unterschiedliche Empfänger gebunden werden.

```
interface IMonitorModule : IMainModule {
  public abstract int timeout { public get; public set;
                                default=1; }
  public abstract long next_run { public get; public set;
                                  default=0; }
  public abstract bool running { public get; public set;
                                 default=false; }

  public abstract void setParam(string key, string value);
  public abstract string getParam(string key);

  public async abstract void monitor();
}
```

Codeblock 5.6: Das Interface eines Überwachungsmoduls besteht hauptsächlich aus der Methode `monitor()`, welche asynchron definiert ist, um nicht auf eine Rückmeldung warten zu müssen. Die Properties `timeout`, `next_run` und `running` dienen zur Prüfung, ob ein Modul aktuell eine Prüfung durchführt oder nicht, sowie um den Zeitpunkt der nächsten Prüfung zu berechnen.

```
interface IDataModule : IMainModule {
  public abstract IDataModule.DataType data_type { public get;
                                                   public set; };
  public abstract bool open(string db_name);
  public abstract bool select(string? fields="*",
                              string? groupBy="");
  public abstract void setWhere(string field, string value);
  public abstract bool hasNext();
  public abstract IDataModule.DataSet[] getCurrent();
  public abstract string getValue(string field);
  public abstract bool save(IDataModule.DataSet[] values,
                            IDataModule.DataSet[]? where=null);
  public abstract bool delete(IDataModule.DataSet[] where);
  public abstract bool empty();

  public struct DataSet { ... };
  public struct SchemaField { ... };
  public enum DataType { ... };
  public enum SchemaFieldType { ... };

  public const string TABLE_XX = "Tablename";
  public const string[] SCHEMA_XX = {
      "name,type,length,defval,primkey",
      "..."
  };
}
```

Codeblock 5.7: Das Interface eines Datenmodules beinhaltet Methoden zum Auslesen wie auch zum Schreiben von Daten. Zusätzlich sind noch verschiedene Datentypen, Konstanten und Strukturen definiert, welche bei der Initialisierung und der Datenbehandlung gebraucht werden. Die zusätzlichen Konstanten, Strukturen und Datentypen werden hier nur exemplarisch dargestellt.

```
interface IAlertModule : IMainModule {
  public signal void onSent(string[] recp, string uid,
                            int state, string? msg="");
  public abstract void setParam(string key, string value);
  public abstract async void send(string subject,
                                  string short_message,
                                  string message,
                                  string[]? attach=null);
}
```

Codeblock 5.8: Das Interface eines Alert-Modules ist sehr einfach und besitzt eigentlich nur eine Methode um einen Alert zu senden, ein Signal für die Übermittlungsbestätigung und eine Methode zur Konfiguration des Modules.

```
1: interface IRemoteModule : IMainModule {
2:   public abstract void startThread();
3:   public abstract void setParam(string key, string value);
4:   public abstract void parseRequest(string? data="");
5: }
```

Codeblock 5.9: Ein Remote-API Modul muss lediglich bei der Instanzierung konfiguriert werden. Anschliessend wird über die Methode `startThread()` ein eigener Thread gestartet, welcher die komplette Kommunikation mit den Clients übernimmt. Der Konformität zuliebe wurde die Methode `parseRequest(string? data="")` dennoch in das Interface aufgenommen.

Kommando	Daten	Beschreibung
discover	`IP-ADDRESS MASK`	Startet die automatische Suche nach Systemen im angegebenen Netzwerk.
testdata	`IP-ADDRESS MASK NUM`	Erstellt `NUM` Pseudo-Systeme in der Datenbank im angegebenen Netzwerk für verschiedene Tests.
testdata	-	Ohne Daten werden alle Pseudo-Systeme aus der Netzwerk- und der Hosts-Tabelle entfernt, nicht jedoch aus dem Monitoring.
monitor	`IP-ADDRESS[ ...]`	Teilt dem System mit, dass es für die Überwachung aller angegebenen IP-Adressen zuständig ist.
synchronize	-	Ohne Daten wird die Synchronisation gestartet.
synchronize	`netw (\t A.B.C.D/MASK)+`	Durch das Prefix „netw“ sowie der Angabe aller Netzwerke als Tabulator getrennte Liste, werden alle Netzwerke aus der Datenbank durch die angegebenen ersetzt.
synchronize	`host \t IP (\t MOD;KEY;VAL)+`	Durch das Prefix „host“ sowie der Angabe einer IP-Adresse werden alle Hosts-Einträge aus der Datenbank entfernt. Die Tabulator getrennte Liste, bestehend aus den Angaben „ModuleName;Key-Name;Wert“, anschliessend verwendet um neue Datensätze zu generieren.
alert	`MODULE IP-ADDRESS GUID`	Teilt dem System mit, dass das gegebene Modul bei dem System fehlgeschlagen und ein Alert mit der GUID versendet worden ist.
rebuild	-	Verknüpft alle Systeme zu einem neuen Überwachungs-Netzwerk.
exit	-	Beendet das System.

Tabelle 5.1: Kommandos und Daten, die ein entferntes System senden kann.

Feld	Typ	Länge	Std.	PK	Beschreibung
section	string	20	-	No	Bereich des Parameter
key	string	50	-	No	Parameter Name
value	text	-	-	No	Parameter Wert

Tabelle 5.2: Datenbank `CONF`: Konfiguration verschiedener Systembereiche.

Feld	Typ	Länge	Std.	PK	Beschreibung
network	string	39	-	No	Netzwerk (IP-)Adresse
mask	int	2	0	No	Subnetmaske
discoverable	int	1	0	No	Das Netzwerk kann untersucht werden 1=ja, 0=nein
vpn_gateway	string	39	-	No	Adresse des VPN-Gateway in dieses Netzwerk
vpn_user	string	20	-	No	Benutzer für die OpenVPN-Verbindung
vpn_pass	string	20	-	No	Passwort für die OpenVPN-Verbindung
is_test	int	1	0	No	Ist auf 1, wenn es sich um ein Pseudo-Test-Netzwerk handelt

Tabelle 5.3: Datenbank `NETWORKS`: Definition der verschiedenen Netzwerke, die überwacht werden.

Feld	Typ	Länge	Std.	PK	Beschreibung
host	string	39	-	No	Adresse des zu überwachenden Systems
module	string	20	-	No	Modulname
key	string	50	-	No	Parameter-Name
value	text	-	-	No	Parameter-Wert
is_test	int	1	0	No	Ist auf 1, wenn es sich um ein Pseudo-Test-Host handelt

Tabelle 5.4: Datenbank `HOSTS`: Alle Systeme und Überwachngs-Dienste sowie Konfigurationsparameter, welche bei einem Host-Discovery gefunden wurden.

Feld	Typ	Länge	Std.	PK	Beschreibung
host	string	39	-	No	Adresse des zu überwachenden Systems
module	string	20	-	No	Modulname
key	string	50	-	No	Parameter-Name
value	text	-	-	No	Parameter-Wert

Tabelle 5.5: Datenbank `MODULES`: Alle Systeme und Überwachungs-Dienste sowie deren Parameter.

Feld	Typ	Länge	Std.	PK	Beschreibung
host	string	39	-	No	Adresse des zu überwachenden Systems
module	string	20	-	No	Modulname
state	uint	4	0	No	Status
time	uint	11	0	No	Timestamp
message	text	-	-	No	Nachricht/Daten

Tabelle 5.6: Datenbank `MONITOR`: Alle Monitoring-Nachrichten.

Feld	Typ	Länge	Std.	PK	Beschreibung
host	string	39	-	No	Adresse des zu überwachenden Systems
module	string	20	-	No	Modulname
subject	string	20	-	No	Betreff der Nachricht
short	string	160	-	No	Kurze Meldung
message	text	-	-	No	Ausführliche Meldung
uid	string	64	-	No	Die GUID des Alerts

Tabelle 5.7: Datenbank `ALERT`: Alle Alert-Nachrichten, die durch das aktuelle System versandt worden wären. Es werden auch die Nachrichten gespeichert, welche aufgrund des Timeouts nicht versendet worden sind.

Feld	Typ	Länge	Std.	PK	Beschreibung
host	string	39	-	No	Adresse des zu überwachenden Systems
module	string	20	-	No	Modulname
tstamp	uint	11	0	No	Zeitstempel des aktuellen Systems
sent_host	string	39	-	No	Adresse des Systems, welches die Nachricht versandt hat
uid	string	64	-	No	Die GUID des Alerts

Tabelle 5.8: Datenbank `ALERT_SENT`: Alle Alert-Nachrichten, welche von anderen und diesem System versandt worden sind. Dient zur Prüfung, ob eine Benachrichtigung versendet werden soll oder nicht.

Key	Value	Beschreibung
to_list	em@il.dom,em@il.dom	Kommaseparierte Liste mit allen EMail-Adressen, an welche ein Alert versendet werden soll
mail_from	em@il.dom	EMail-Adresse des Alert-Absenders
host	IP-Address | DNS	IP-Adresse oder DNS-Name des MX-Servers
auth_method	plain | login	Authentifizierungs-Methode. Aktuell werden LOGIN und PLAIN unterstützt
username	em@ail.com	Benutzername für Relaying
password	abc	Passwort zum obigen Benutzernamen

Tabelle 5.9: Die Datei „dummy_alert_config.txt“ im data-Ordner beinhaltet aktuell die Konfiguration des EMail-Alert-Modules. Key und value werden durch einen Doppelpunkt getrennt: „host:1.2.3.4“.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich diese Thesis selbständig verfasst und keine andern als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche kenntlich gemacht. Ich versichere zudem, dass ich bisher noch keine wissenschaftliche Arbeit mit gleichem oder ähnlichem Inhalt an der Fernfachhochschule Schweiz oder an einer anderen Hochschule eingereicht habe. Mir ist bekannt, dass andernfalls die Fernfachhochschule Schweiz zum Entzug des aufgrund dieser Thesis verliehenen Titels berechtigt ist.

Lommis, 09. Februar 2011