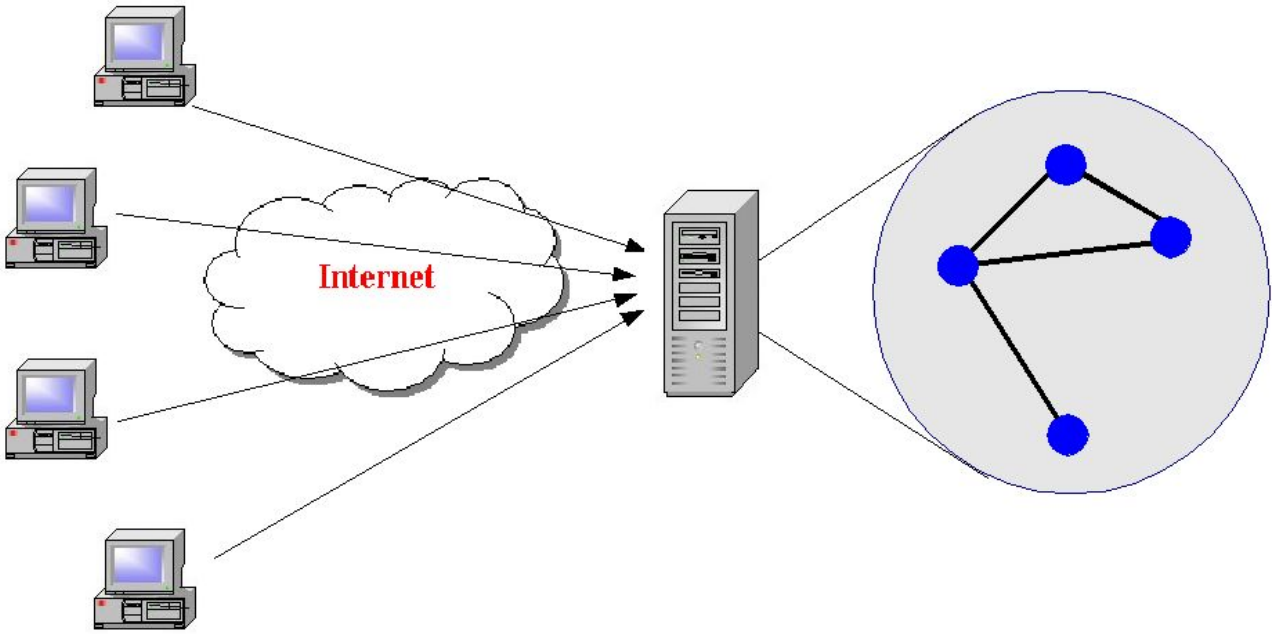
VPN Network Testbed

Dokumentation und Installationshandbuch



Semesterarbeit

René Gallati <devel (AT) gallati.net>

Betreuer

Nicolas Burri

Inhaltsverzeichnis

Übersicht

Das VPNServer-System ist ein aus mehreren Komponenten aufgebautes System, welches es durch den geschickten Einsatz verschiedener Netzwerktechnologien ermöglicht, ein vollwertiges Netzwerk auf Ethernetbasis zu simulieren. Dabei laufen die einzelnen Knoten auf unterschiedlichen Systemen, wobei auf einem Rechner auch mehrere Knoten laufen können. Die Verbindung zwischen den Knoten und dem steuernden Server wird mittels einer leistungsfähigen VPN Technologie hergestellt. Auf diese Art und Weise kann auch ein grösseres Netzwerk mit mehreren verteilten Rechnern simuliert werden, da nicht ein einzelner Rechner die gesamte Last tragen muss.

Grundsätzlich verbinden sich die Knoten über die VPN-Software mit dem VPNServer. Damit wird auf beiden Seiten der VPN-Verbindung eine neue virtuelle Netzwerkkarte, ein sogenanntes „TAP-Device“, erzeugt. Dieses erhält nun eine IP Adresse aus einem der privaten IP-Adressbereiche[1] zugewiesen. Auf der Serverseite wird die virtuelle Netzwerkkarte mitteles einer (Software-)Bridge mit den anderen Knoten zusammengeschalten. Damit formen diese Knoten dann über den VPNServer ein virtuelles Netzwerk, welches über die Firewall des VPNServers gesteuert wird, so dass eine spezifische Netzwerktopologie aufgebaut werden kann.

Systemübersicht

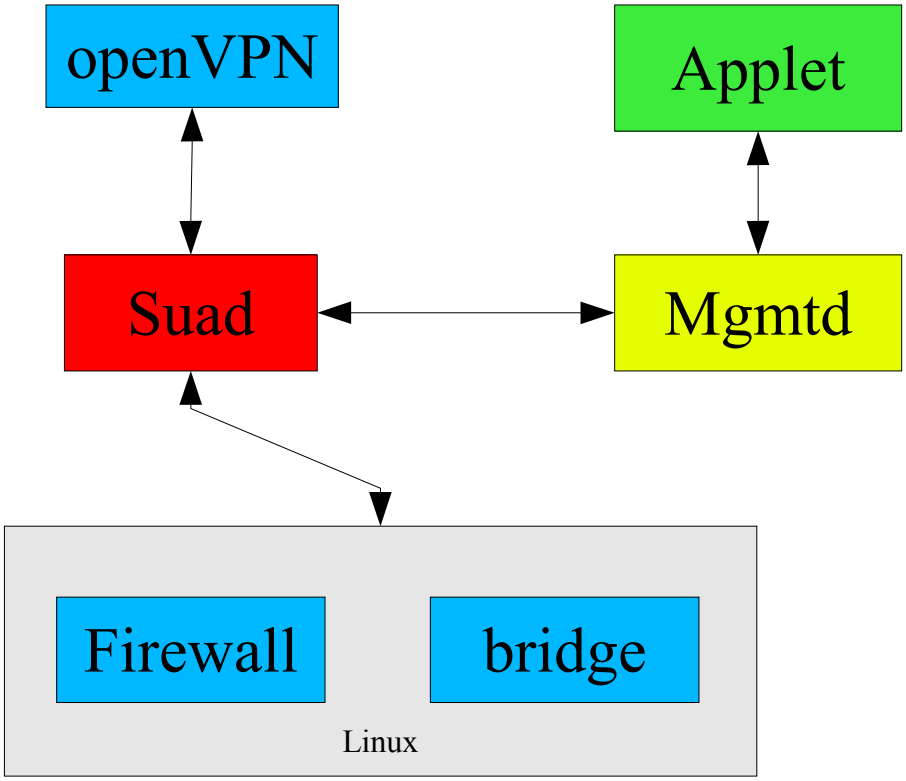


Abbildung 1 VPNServer Komponenten

1 Siehe RFC 1918

Systemkomponenten

Der VPNServer selber besteht aus mehreren Teilsystemen mit unterschiedlichen Aufgaben. Als Basis dient ein Standard-Linux Betriebssystem dessen leistungsfähiges Netzwerksystem zusammen mit der erstklassigen Netfilter-Firewall das solide Fundament bildet.

Die Verbindung zwischen dem Server und den Knoten wird über die von openVPN aufgebauten Tunneln hergestellt. OpenVPN ist eine frei erhältliche Software welche im Gegensatz zu vielen anderen VPN-Softwarelösungen auch problemlos parallel zu anderer VPN-Software eingesetzt werden kann, wie es häufig in Firmenumgebungen, aber auch zum Beispiel für Laptop-Docking und/oder WLAN wie an der ETH eingesetzt wird. Herkömmliche VPN-Lösungen dulden in der Regel keine andere VPN-Software neben sich und haben aus Sicherheitsgründen oft auch sehr starke Restriktionen zur Folge. OpenVPN ist eine kleine und übersichtliche, aber dennoch sichere VPN-Lösung auf SSL-Basis welche nicht nur einen reinen IP-Tunnel (IP zu IP oder IP-Netz) aufbauen, sondern eine komplette virtuelle Netzwerkkarte simulieren kann. Darüber besteht dann die Möglichkeit, nicht nur IP, sondern alle gültigen, auf Layer 2[2] befindlichen Netzwerkprotokolle über das VPN zu transportieren. Damit sind effektiv die gleichen Fähigkeiten wie in einem lokalen Ethernet-LAN möglich, insbesondere normalerweise für VPNs nicht transportable Protokolle wie Ethernet-Broadcasts, Multicast, ARP, IGMP, IPv6, etc.

Die Steuerung der Hauptfunktionalität des VPNServers übernimmt ein in dieser Semesterarbeit geschriebenes Java-Programm namens Suad (Suad steht für SuperUserAccessDaemon). Der Name weist darauf hin, dass dieses Java Programm unter dem root-Account läuft und deshalb vollständige Systemrechte besitzt. Der Suad aktiviert und kontrolliert die anderen Teilssysteme. Er startet und verwaltet die openVPN Instanzen, von welchen es eine pro bekanntem User (Knoten) gibt. Weiterhin steuert er die Firewall, was ebenfalls root-Rechte voraussetzt und startet den Mgmtd.

Der Mgmtd, was für ManagementDaemon steht, ist ein weiteres Java-Programm, welches für diese Semesterarbeit geschrieben worden ist. Dieses verwaltet die sogenannten Strategie-Module, welche das Layout des virtuellen Netzwerkes kontrollieren. Da diese Strategie-Module als externe Plugins, welche in ein spezielles Verzeichnis kopiert werden können, angesehen werden, läuft der Mgmtd als separates Programm mit nicht-privilegierten Rechten. So kann ein Fehler in einem Strategie-Modul keine schwerwiegende Auswirkungen auf das Serversystem haben und die Systemsicherheit kann nicht gefährdet werden. Der Mgmtd verwaltet ausserdem die Layout-Konfiguration, so dass unterschiedliche Layouts für eine unterschiedliche Anzahl von verbundenen Knoten eingestellt werden können. Schlussendlich kommuniziert der Mgmtd mit dem Applet, über welches Benutzer ihren Account erstellen, sowie das momentane Netzwerklayout grafisch anschauen können.

Das Applet auf dem Webserver schlussendlich kommuniziert mit dem Mgmtd um Benutzern die Accountdaten für die VPN Verbindung zugänglich zu machen, sowie das aktuelle Netzwerklayout anzuzeigen. Zusätzlich kann der Administrator über das Applet einige Konfigurationsänderungen durchführen, wie beispielsweise das Netzwerklayout manuell verändern oder die Reihenfolge der Strategie-Module neu einstellen.

2 Layer2 Datenübertragungsschicht gemäss dem OSI Schichtenmodell

Das Applet

Das Applet stellt die Schnittstelle für die Benutzerkonten dar. Da jede VPN-Verbindung über einen eigenen Schlüssel sowie jeder VPN-Tunnel über eine eigene IP-Adresse und einen separaten UDP-Port zum Verbinden verfügen muss, benötigt man pro Netzwerkknoten ein Benutzerkonto. Diese Konti können bequem über das auf dem Webserver des VPNServers laufende Applet erstellt oder die Konfigurationsdaten später wieder abgerufen werden.

Applet:

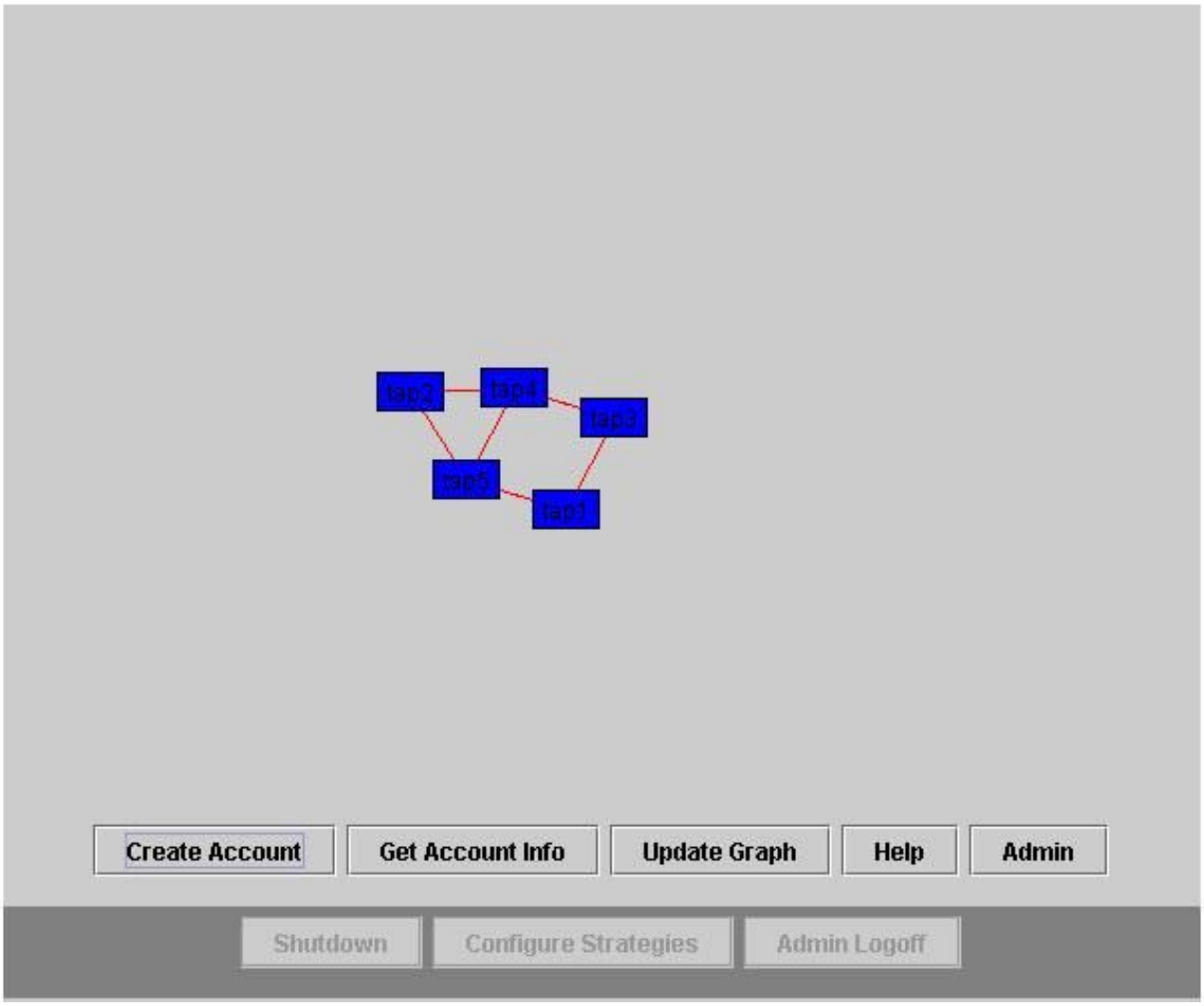


Abbildung 2 Das Applet

Über das Applet können Benutzerkonten erstellt („Create Account“) und die entsprechenden Konfigurationsdaten später nochmals eingesehen werden („Get Account Info“). Über den „Update Graph“ Button kann die aktuelle Netzwerktopologie angezeigt werden. Durch einen Klick auf den „Help“ Button verzweigt der Browser auf Hilfsseiten, welche die Benutzung des Applets und die Installation von openVPN auf den Client-Rechnern genauer erklären.

Mittels des „Admin“ Knopfes schlussendlich werden nach Eingabe des Administratorpasswortes die unteren Buttons aktiviert. Zur Anzeige dass der Admin-Modus aktiv ist, färbt sich die Buttonleiste blau. Der Admin-Modus ermöglicht eine Rekonfiguration der Strategie-Module wie auch das direkte „malen“ von Verbindungen, indem in der Grafik mittels rechter Maustaste eine Verbindung zwischen verschiedenen Knoten gezeichnet wird. Um Verbindungen zu entfernen, zieht man die Linie entlang zwei schon verbundener Knoten nach.
Der „Shutdown“ Knopf beendet sofort das VPNServer System, wobei es wegen des im Hintergrund laufenden keepalived.pl automatisch nach einer Minute wieder gestartet wird. Für Details siehe das Kapitel Starten und Stoppen des VPNServers.

Um den Admin-Modus zu verlassen, genügt ein Klick auf die „Admin Logoff“ Schaltfläche, woraufhin sich die Buttonleiste wieder grau färbt und die Admin-Knöpfe nicht mehr klickbar geschaltet werden.

Codeübersicht

Die beiden Java-Komponenten (oder drei, wenn man das Applet mitzählt) bestehen aus insgesamt 34 Java-Klassen. Damit ein Einstieg in den Code einfacher erfolgen kann, werden kurz die wichtigsten Klassen aufgeführt bevor ein exemplarischer Codelauf erklärt wird. Grundsätzlich sind alle Klassen relativ ausführlich mit Javadoc Kommentaren angereichert, welche ein zurechtfinden stark erleichtern sollten.

Wichtige Klassen

ch.ethz.dcg.vpnsrv.*CommandList*: *CommandList* ist eine Klasse, welche eine Serie von ShellKommandos aufnehmen kann und diese dann auch ausführt, inklusive dem notwendigen Flush von StandardError und StandardOut. Mittels *CommandList* kann man also eine beliebige Anzahl von Befehlen so ausführen, als würde man sie auf einer Linux Konsole oder Shell eingeben. Dies wird verwendet, um die Firewall zu rekonfigurieren wie auch um neue openVPN Instanzen oder den Mgmtd als nicht-privilegierten Benutzer zu starten.

ch.ethz.dcg.vpnsrv.*Mgmtd*: Der ManagementDaemon ist eine der beiden wichtigsten Komponenten im System. Der *Mgmtd* startet eigene Threads, verbindet sich zum *Suad* über einen Socket und tauscht mit diesem Daten aus. Er startet und verwaltet weiterhin die Strategie-Module und kommuniziert mit den Applets.

ch.ethz.dcg.vpnsrv.*StrategyLoader*: Eigener Classloader der die Strategie-Module finden und laden kann.

ch.ethz.dcg.vpnsrv.*StrategyManager*: Komponente, welche die Verwaltung der Strategie-Module übernimmt und dem *Mgmtd* mitteilen kann, bei welcher Anzahl an Knoten welche Strategie aktiv sein soll.

ch.ethz.dcg.vpnsrv.*Suad*: Der SuperUserAccessDaemon – die wichtigste Komponente im System. Der *Suad* läuft unter dem root-Account und hat daher Zugriff auf das komplette System. Er steuert die Firewall, startet oder beendet openVPN Instanzen und rekonfiguriert die Netzwerk-Bridge wenn neue Knoten hinzukommen oder alte entfernt werden. Weiterhin beeinhaltet er die gesamte Benutzerverwaltung und hält auch eine *NetMap*-Instanz, welche jederzeit der aktuellen Netzwerk-Topologie entspricht.

ch.ethz.dcg.vpnsrv.*UserManager*: Der UserManager verwaltet die Dateien (4 pro Benutzer) in einem separaten Verzeichnis und ist auch in der Lage neue Konfigurationen für neue Benutzerkonten zu erzeugen. Er verwaltet weiterhin den „count“, die Anzahl der bereits registrierten Benutzer, was indirekt die nächste zu vergebene IP und Port Adressen beeinflusst.

ch.ethz.dcg.vpnsrv.applet.*: Alle Klassen welche das Applet und dessen Dialoge darstellen.

ch.ethz.dcg.vpnsrv.data.*AppletMessage*: Die Nachrichten vom Applet werden in dieser Klasse gekapselt und verwaltet. *AppletMessages* werden in *EventMessages*, welche prinzipiell gleich aufgebaut sind,

gekapselt.

ch.ethz.dcg.vpnsrv.data.*EventMessage*: Nachrichten zwischen Java-Komponenten des Systems werden immer in *EventMessage* Instanzen verpackt. Diese können sich selber in einen Byte-Datenstrom serialisieren und somit einfach versandt und behandelt werden. Sämtliche über TCP ablaufende Kommunikation wird durch die EventMessages abgewickelt. Sie stellen sozusagen das Netzwerkprotokoll dar.

ch.ethz.dcg.vpnsrv.data.*NetMap*: In der *NetMap* Klasse befindet sich der Netzwerk-Topologiegraph sowie Methoden zur Manipulation desselben und ein Mechanismus um automatisch bei Ereignissen benachrichtigt zu werden.

ch.ethz.dcg.vpnsrv.util.*ByteArrayIO*:Klasse zur einfachen Serialisierung und Deserialisierung von Basistypen in einen Bytestrom. Wird primär für die Konvertierung von Datenstrukturen in ein ByteArray und umgekehrt verwendet, um die langsame Java-Serialisierung durch Introspektion und Reflektion zu umgehen.

Beispiel des Programmflusses durch das System

Als Beispiel für den Programmfluss durch das System dient ein sich mit dem System verbindender Benutzer, der seinen openVPN Tunnel aufbaut. Dieses Beispiel wurde gewählt, da dieses Ereignis an den Suad gemeldet wird, der es aber an den Mgmtd weiterreichen und dessen Antwort auswerten muss. In der folgenden Ablaufskizzierung sind weniger wichtige Hilfsklassen der Einfachheit halber weggelassen worden:

1. Ein Benutzer startet seine openVPN Software welche eine UDP-Verbindung zum Server herstellt und das VPN aufbaut. Sobald das VPN zwischen den beiden Systemen steht, ruft openVPN auf dem VPNServer das shell-skript `/opt/vpnsrv/bin/up` auf, welches eine lokale Telnet Verbindung auf den von einem *NotifyChild* kontrollierten TCP-Socket öffnet und darüber durch ein simples „UP tap12“ die *Suad* Komponente von der neuen Verbindung in Kenntnis setzt.
2. Diese Nachricht über den Socket wird von der Methode *Suad*.handleInterfaceUp(String) entgegengenommen, welche den Parameter (den Device-Namen des Tap-Devices) aus dem String ausliest und ein neues *Node*-Objekt, das diesem Knoten mit dem entsprechenden Tap-Namen geniert, erzeugt. Dieses *Node*-Objekt fügt der Suad nun seiner lokalen *NetMap* Instanz, welche der Netzwerk-Topologie entspricht, mittels netmap.addNode(node) hinzu.
3. Die *NetMap* prüft zunächst ob sich der entsprechende Knoten schon in der Topologie befindet und macht gar nichts, falls dies so sein sollte. Ist der Knoten hingegen neu, so fügt NetMap diesen zur lokalen Liste hinzu und generiert eine *EventMessage* welche diesen neuen Knoten ankündet. Diese Nachricht wird nun über den in *NetMap* eingebauten Mechanismus an alle für den Empfang solcher Nachrichten registrierten Komponenten gesandt. Einer dieser *EventListener* ist *MgmtChild*, eine Klasse welche in ihrem eigenen Thread läuft und sich exklusiv um die Kommunikation mit dem *Mgmtd* kümmert.
4. Der *MgmtChild*-Thread erhält nun die Nachricht über den neuen Knoten von *NetMap* in *MgmtChild*.newEvent(EventMessage) und sendet dieses Ereignis unverändert über *MgmtChild*.sendEvent(EvenetMessage) an den *Mgmtd* über den TCP-Socket.
5. Auf Seiten des *Mgmtd*'s kümmert sich die ebenfalls in einem eigenen Thread laufende Klasse *SuadConnector* um die Kommunikation. Sie liest in ihrer *SuadConnector*.run() Schlaufe den Socket aus und nimmt die Nachricht entgegen, welche sie anschliessend in der

SuadConnector.processMessage(EventMessage) Methode verabeitet. Dort wird festgestellt, dass es sich um ein NODE_ADDED Ereignis-Nachricht handelt und ein *Node* Objekt wird auch hier erzeugt und der lokalen, also dem *Mgmtd*-gehörigen *NetMap* Instanz hinzugefügt.

6. Die dem *Mgmtd* gehörige *NetMap* Instanz macht nun genau dasselbe wie diejenige von *Suad* in Schritt 3, mit dem Unterschied dass hier die Benachrichtigung von *Mgmtd* nicht über den eingebautet Nachrichten-Dispatch Mechanismus von *NetMap* benutzt wird, sondern explizit *Mgmtd*.newNode(Node) aufgerufen wird.
7. In *Mgmtd*.newNode(Node) wird die *Mgmtd*.activateStrategy() Methode aufgerufen, welche über den *StrategyManager* prüft, ob das aktuelle Strategie-Modul gegen ein anderes ausgetauscht werden muss, da nun ein Knoten mehr da ist, oder ob das momentan noch geladene weiterhin zuständig ist. Danach wird über *Mgmtd*._activeStrategy.nodeAdded(Node) das momentan zuständige Strategie-Modul über den neuen Knoten informiert.
8. Das Strategie-Modul kann nun – je nach Zweck – Kanten entfernen und hinzufügen. Exemplarisch soll das hinzufügen einer Kante angeschaut werden.
9. Das Strategie-Modul fügt über *NetMap*.addConnection(Node, Node) eine neue Kante im Topologiegraphen hinzu.
10. *NetMap*.addConnection(Node, Node) überprüft, ob die Kante schon besteht und macht in diesem Fall abermals nichts. Ist die Kante aber neu, so wird sie in die interne Struktur eingetragen und ein CONN_ADDED Ereignis erzeugt, das an alle *EventListeners*, welche an dieser *NetMap* Instanz angemeldet sind, sendet. *Mgmtd* ist ein solcher Listener und die CONN_ADDED *EventMessage* landet somit bei *Mgmtd*.newEvent(EventMessage).
11. In *Mgmtd*.newEvent(EventMessage) wird erkannt, dass es sich um eine CONN_ADDED *EventMessage* handelt und diese wird sogleich über den *SuadConnecter*.sendEvent (EventMessage) weiterversandt.
12. *SuadConnector*.sendEvent schreibt die EventMessage auf den Socket, sie erscheint somit am von *MgmtChild* kontrollierten Socket-Endpunkt.
13. *MgmtChild*.run() liest die Bytes vom Socket und regeneriert daraus wiederum ein *EventMessage* Objekt, welches an die interne *MgmtChild*.processMessage(EventMessage) Methode weiterreicht. Dort wird das Ereignis als CONN_ADDED erkannt und sogleich an *Suad*.newEvent(EventMessage) weitergereicht.
14. In *Suad*.newEvent(EventMessage) wird *Suad*.addConnection(String) aufgerufen, was dafür sorgt, dass die neue Kante nun auch in der lokalen *NetMap* Instanz mittels *NetMap*.addConnection(Node, Node) eingetragen wird. Somit sind nun die beiden *NetMap* Instanzen auf Seiten *Suad* und *Mgmtd* wieder identisch. Durch das Eintragen der Verbindung wurde aber natürlich wieder eine *EventMessage* erzeugt, welche an *Suad*.newEvent (EventMessage) gesandt wird.
15. Im Gegensatz zu *EventMessages*, welche über den TCP Socket versendet werden, haben lokal generierte *EventMessages* aber keine ID-Nummer, respektive tragen dort den Wert -1. Dadurch wird in *Suad*.newEvent(EventMessage) diese Nachricht anders als in Schritt 13 behandelt und aus der Nachricht eine *CommandList* generiert. Diese Fähigkeit haben einige der *EventMessages;* sie können aus den mitübermittelten Parametern gleich eine Sequenz an Kommandos erzeugen, die sie dann durch *EventMessage*.createCommands(String) zurückliefern.
16. Diese *CommandList* wird jetzt, immer noch in *Suad*.newEvent(EventMessage) durch *CommandList*.execute() aufgerufen.
17. *CommandList*.execute führt die Kommandos aus und ändert nun die Firewall auf dem VPNServer, so dass die beiden Knoten, welche über diese Kante verbunden sind, nun auch im simulierten Netzwerk miteinander kommunizieren können.
18. Schlussendlich ist die in Schritt 2 in *Suad*.handleInterfaceUp(String) begonnene Reise durch das System beendet und es wird noch in *Suad*.logUserOn(String) über den *UserManager* festgestellt, welcher UserAccount diese Verbindung initiert hat und ein entsprechender Logeintrag im User-Log erstellt.

Bemerkungen zur Ablaufskizze

Wie man der Ablaufskizze entnehmen kann, spielen EventMessages eine zentrale Rolle. Sie werden durch Änderungen an einer NetMap generiert und automatisch an alle dort registrierten, interessierten EventListener gesandt oder aber auch explizit erzeugt und durch das System versandt. Verarbeitet und aussortiert werden sie in den beteiligten Klassen grundsätzlich in newEvent (EventMessage) Methoden, wie es das EventListener-Interface vorschreibt – oder aber in processMessage(EventMessage) sofern die EventMessage über einen TCP-Socket empfangen wurde.
Aus EventMessages können dann, wie im Schritt 15 aufgezeigt, direkt CommandLists erzeugt werden, welche die eigentliche Konfigurationsänderung am System vornehmen. Es ist von daher anzuraten, die EventMessage Klasse und deren Interaktion mit dem System genauer anzusehen. Um dem VPNServer erweiterte Funktionalität hinzuzufügen recht es nämlich in vielen Fällen aus, nur einen neuen EventMessage-Typen zu konstruieren und an den richtigen Ort zu senden (in der Regel an den Suad).
Wenn man den Nachrichtenfluss im laufenden System genauer verfolgen möchte, reicht es aus, die jeweiligen Log-Levels der Subsysteme in /opt/vpnsrv/config/logging.conf auf DEBUG zu stellen, so dass die processMessage(EventMessage) Methoden den Empfang von EventMessages und deren Typ und Informationen ins Log schreiben.

Installation

Die Installation des VPNServers gliedert sich in 2 Teile: Installation des Linuxsystems und die Installation und konfiguration der VPNServer-Komponenten

Vor der Installation

Benötigt wird folgendes:

1. Ein PC („Server“) der i386-kompatiblen Art auf welchem die Software installiert wird. Dieser Server sollte mindestens einen 1GHz Prozessor und 256 MB Ram haben. Je nach Grösse der zu simulierenden Netzwerke muss die CPU leistungsfähiger resp. mehr Ram vorhanden sein. Der Server benötigt weiter eine Netzwerkkarte nach min. FastEthernet Standard (100mbit) sowie ein CD-Rom oder besser ein DVD-ROM Laufwerk.
2. Die SuSE 9.1 Installations DVD, resp. die SuSE 9.1 Installations CDs sollte der Server kein DVD-Rom Laufwerk besitzen oder ein Installationsimage auf einem erreichbaren FTP/HTTP Server.
3. Die VPNServer Software (.tgz Datei)
4. Die Angaben zur Netzwerk Installation (bei Bedarf beim Netzwerkadministrator anfragen)
5. Dieses Installations- und Bedienungshandbuch zur Installation und Konfiguration der Software

Optionale Komponenten:

- Eine Softwareauswahl-Diskette. Wird diese verwendet, so muss die Softwareauswahl bei der Installation von SuSE Linux nicht manuell durchgeführt werden. Dies erzwingt aber ein Floppy Laufwerk auf dem Server.

Installation

Sind die benötigten Komponenten vorhanden, kann mit der Installation begonnen werden.

Achtung, durch die Installation werden jegliche schon vorhandenen Daten auf dem PC unwiderruflich gelöscht.

Die Installation ist in 2 Teile aufgeteilt. Die Grundinstallation, welche ein SuSE Linux 9.1 System installiert sowie die Installation der VPNServer Software, welche die Grundinstallation voraussetzt.

Die Grundinstallation

1. DVD oder CD #1 in das Laufwerk einlegen und den PC starten. Nach kurzer Zeit sollte der Bootloader Bildschirm wie in Fig.1 dargestellt erscheinen. Die Art der Installation (grafisch / Textmode) kann beliebig über F2 eingestellt werden. Sollen die Kernelmeldungen beim booten angezeigt werden, so sollte unter F5 „verbose“ statt splash stehen. Mittels Cursor rauf/runter kann die Installationsart gewählt werden. Die Auswahl sollte auf „Installation“ wie in Fig.1 dargestellt stehen und kann dann mittels [RETURN] bestätigt werden.

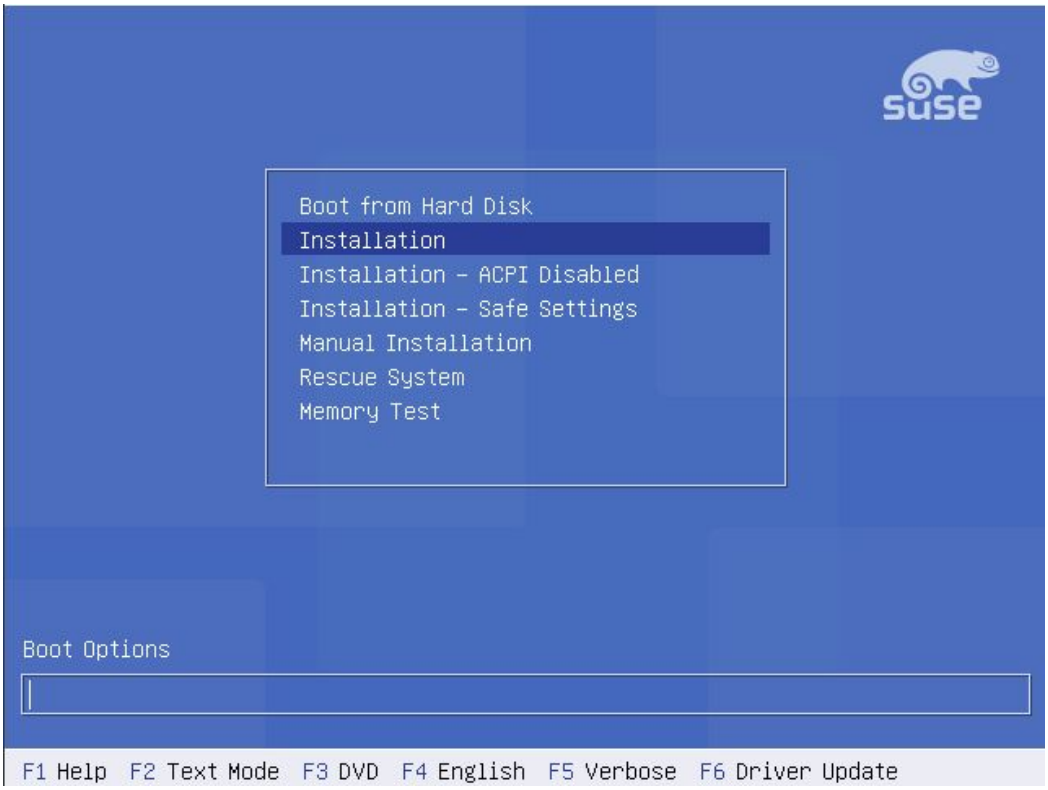


Fig 1 SuSE Linux Bootloader

2. Nach kurzer Zeit erscheinen die Bootmeldungen des Linux Kernels auf dem Bildschirm. Nach einigem weiteren warten erscheint der grafische Installer von YaST (Yet another Setup Tool) sofern eine grafische Installation im ersten Schritt eingestellt wurde.
3. Bei der Auswahl der Sprache wähle **English (US)**. (Fig. 2)

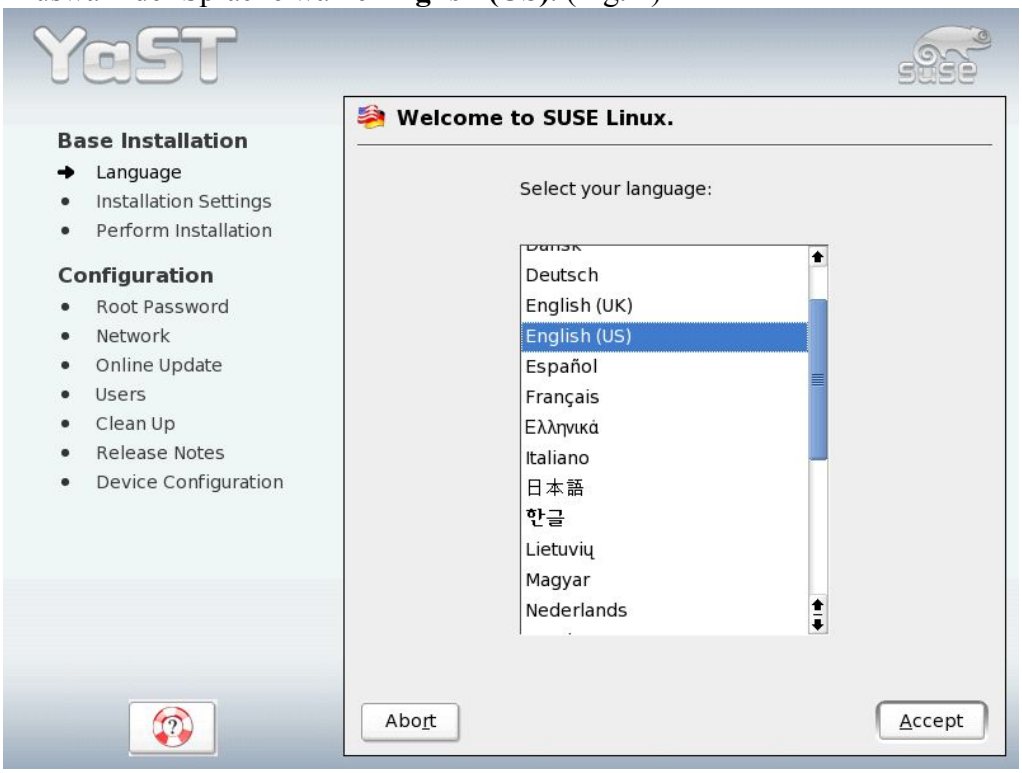


Fig 2 Sprachauswahl

4. Danach untersucht YaST das System und bietet einem die Möglichkeit zur Änderung der Konfiguration (Fig. 3)

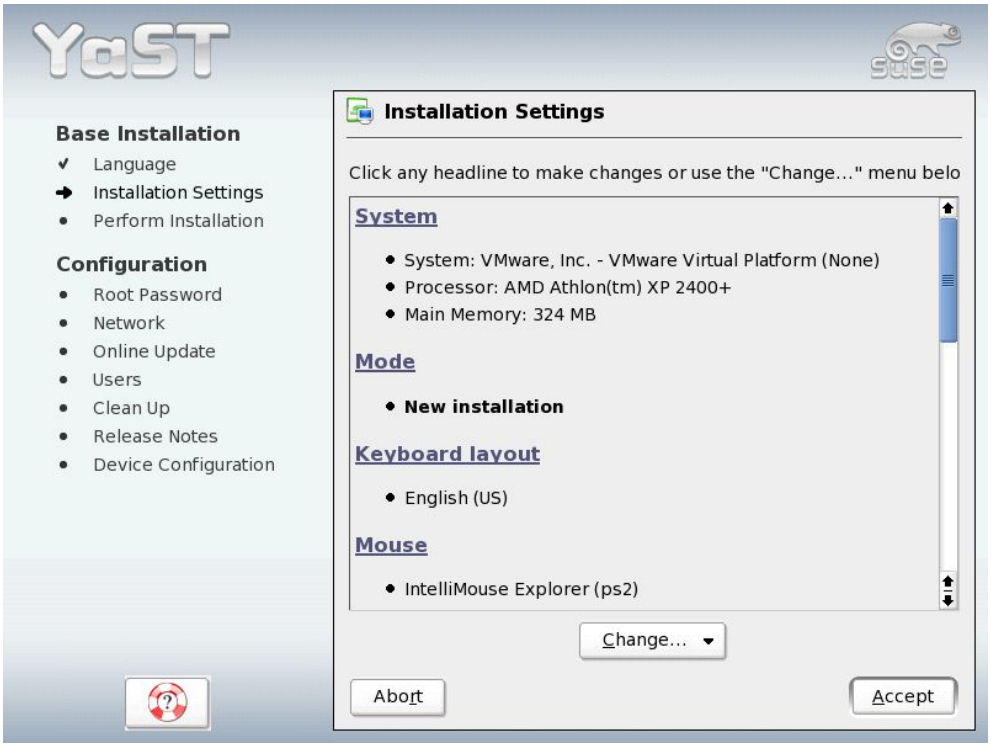


Fig 3 Installationsübersicht

5. Die Tastatureinstellung sollte nun angepasst werden. Dies geschieht durch klicken auf „Change“ sowie „Keyboard Layout“ (Fig. 4)

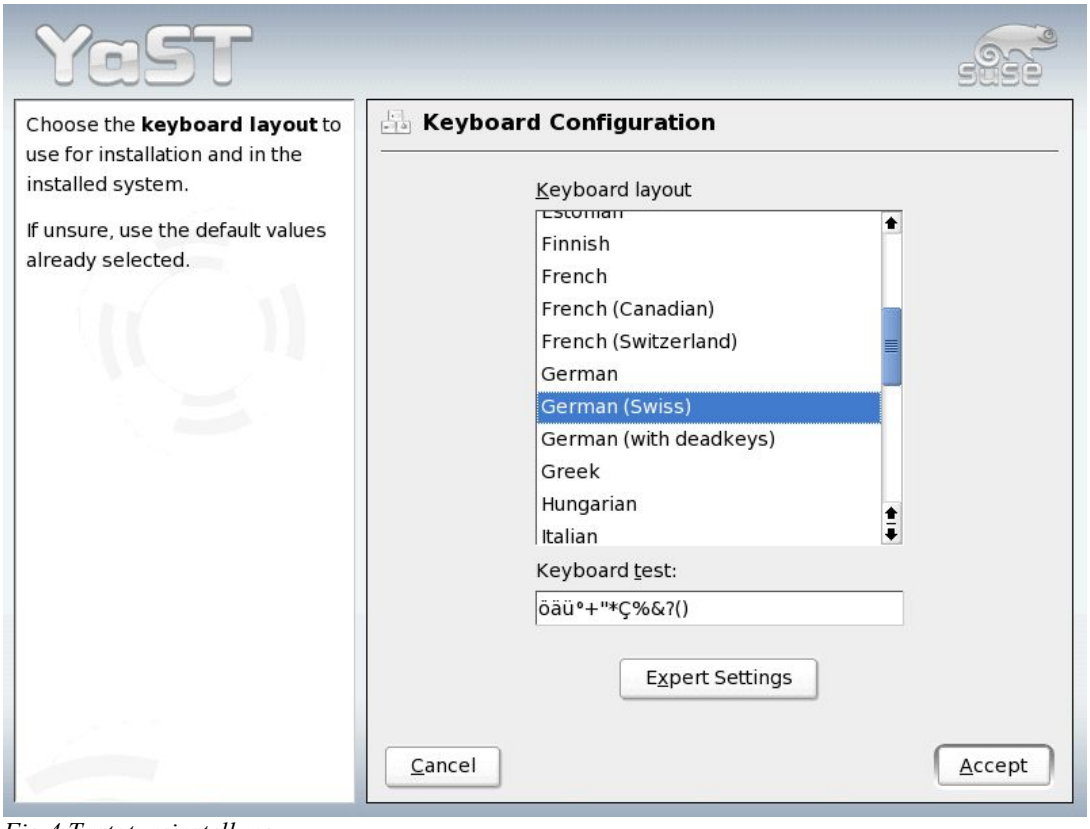


Fig 4 Tastatureinstellung

6. Jetzt kann auch noch die Partitionierung der Festplatte geändert werden. Normalerweise kann der Vorschlag von YaST ungeändert übernommen werden. Sind auf dem PC aber mehrere Betriebssysteme vorhanden muss hier evtl. manuell partitioniert werden. Der VPN Server benötigt nur etwa 700MB Platz auf der Festplatte (zuzüglich einer Swap Partition). 2GB insgesamt sollten ausreichen, wenn der Server nicht noch weitere Aufgaben übernehmen soll.
7. Danach muss die Softwareauswahl getroffen werden. Ist die Softwareauswahl-Diskette vorhanden, so kann diese nun einfach in den Server eingelegt werden und im Menü zur detaillierten Auswahl (Fig.5) „File / Import“ angewählt werden. Hier nun die Diskette auswählen und die Datei (vpnsrv.sel) mittels des Browse-Buttons auswählen.

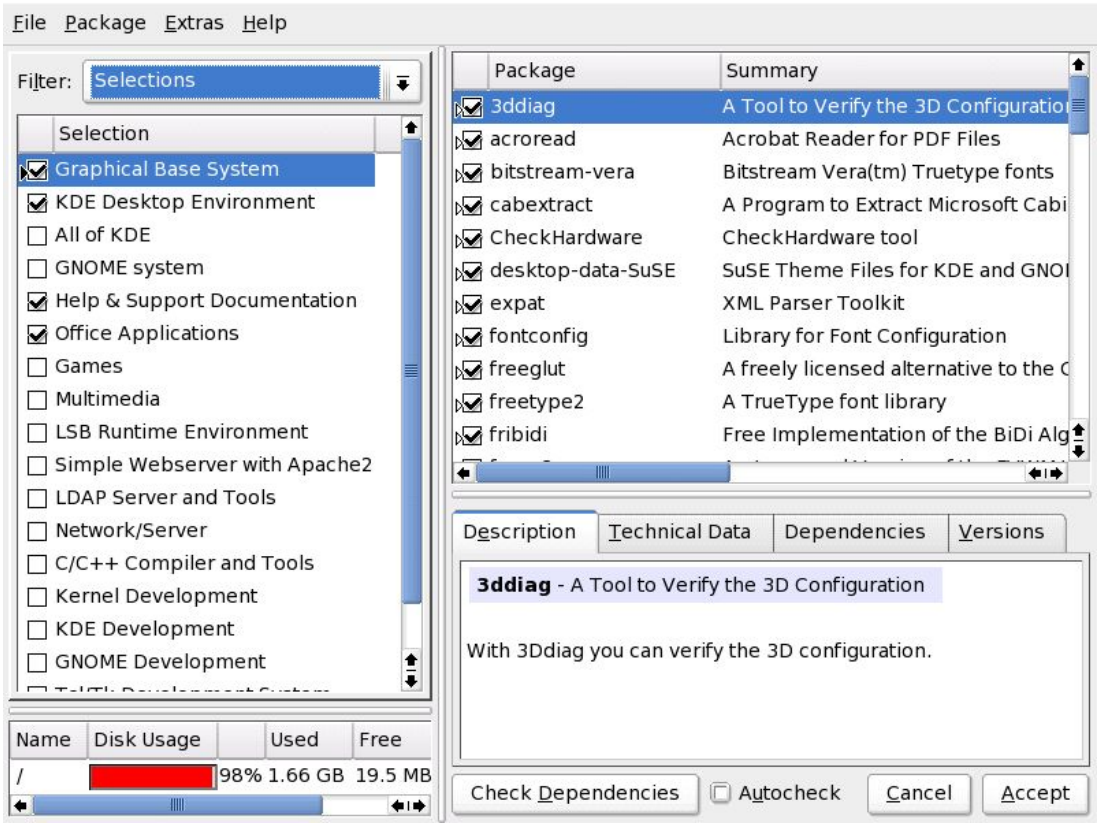


Fig 5 Die Softwarepaketauswahl

Sollte es nicht möglich sein die Softwareauswahl-Diskette zu verwenden (zum Beispiel weil der Server kein Diskettenlaufwerk hat) muss „minimal graphical system“ ausgewählt werden und danach mit einem Klick auf „Detailed Selection“ die im Anhang 5 aufgeführten Softwarepakete zusätzlich ausgewählt werden. Diese können über die Suchfunktionalität schnell in der grossen Auswahl gefunden werden.

8. Als nächstes sollte die Zeitzone auf dem Server eingestellt werden. Eine richtig konfigurierte Zeit ist wichtig, damit die Log-Files sinnvolle Informationen liefern können (Fig. 6)

9. Schlussendlich sollte noch der Default-Runlevel auf 3 gesetzt werden.

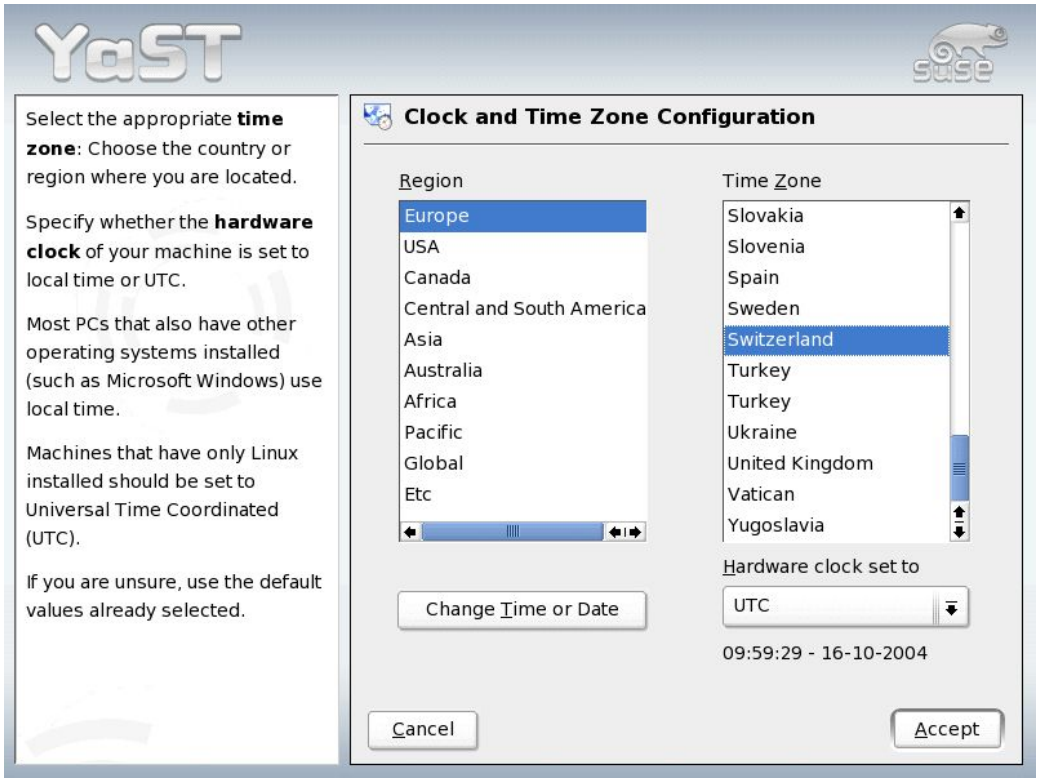


Fig 6 Die Konfiguration der Zeitzone

10. Jetzt kann die Intallation duch einen Klick auf Accept beginnen. YaST fragt noch einmal sicherheitshalber nach, da nach dem Akzeptieren der Dialogbox die Festplatte (um-)partitioniert wird und ein Rückgängigmachen nicht mehr möglich ist.

11. Die Installation beginnt, YaST formatiert die Festplatte und kopiert die Software auf das System, was einige Minuten in Anspruch nehmen kann.Wird das System von CDs und nicht von der DVD installiert, so wird man ggf. aufgefordert die entsprechende CD einzulegen.(Fig. 7)
12. Nach der Softwareinstallation bootet YaST das System automatisch neu. Nach kurzer Zeit erscheint wiederum der Bootloader (Fig.1) wobei man diesmal keine Taste betätigen sollte, so dass das System automatisch neu starten wird.

Fig 7 Die Softwareinstallation des Linuxsystems

13. Nach dem Aufstarten des Systems erscheint wiederum der YaST Installer um noch einige Fragen zur Hardwareinstallation zu stellen. Als erstes wird nach dem root-Passwort gefragt. Dieses sollte möglichst komplex und nicht einfach zu erraten sein und sollte unbedingt aufgeschrieben werden! (Fig.8)

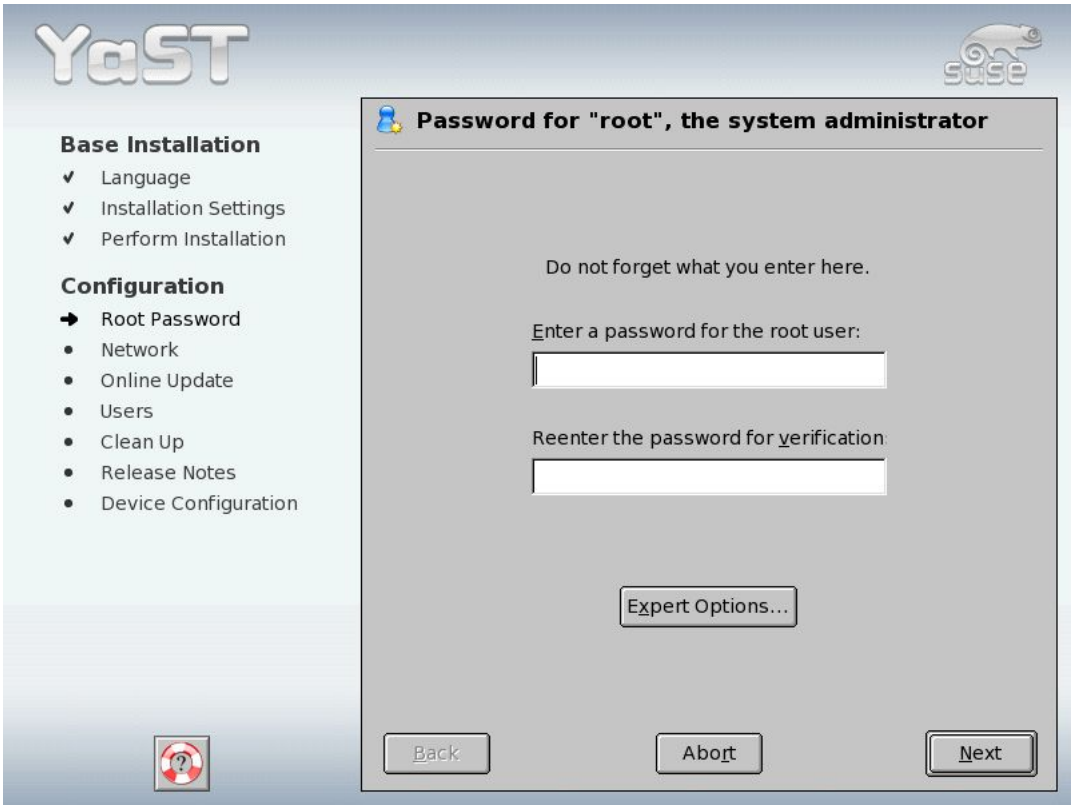


Fig 8 Rootpasswort Dialog

14. Als nächstes folgt die Netzwerkkonfiguration. Hier sollte man die vom Netzwerkadministrator erhaltenen Informationen für die Netzwerkkarte eintragen. Zuerst wählt man dazu den Knopf „Change“ aus und danach „Network Interfaces“ welches einem zur Auswahlmaske für die Netzwerkkarte(n) bringt. Hier kann man mittels change die IP Adresse, Subnetzmaske und den Standardgateway wie auch den Namen des Servers selbst (Vorschlag vpnsrv.dcg.ethz.ch) auswählen (Fig. 9-11)

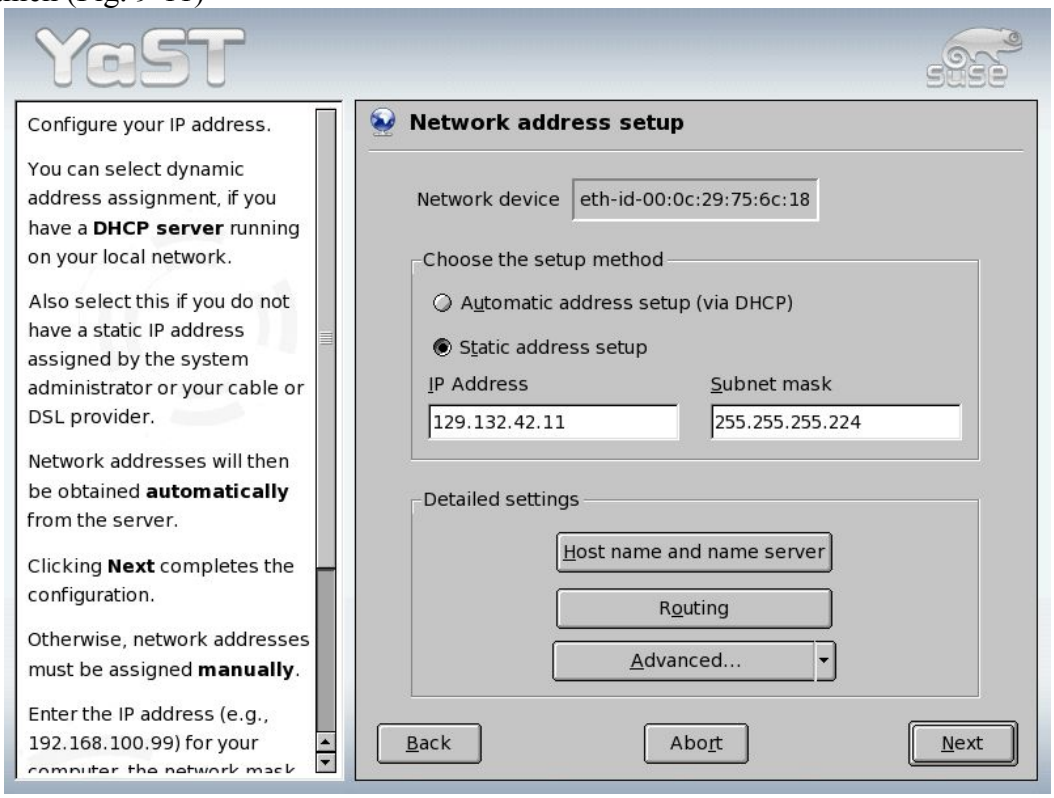


Fig 9 Netzwerkkarten Konfiguration (IP + Netmask)

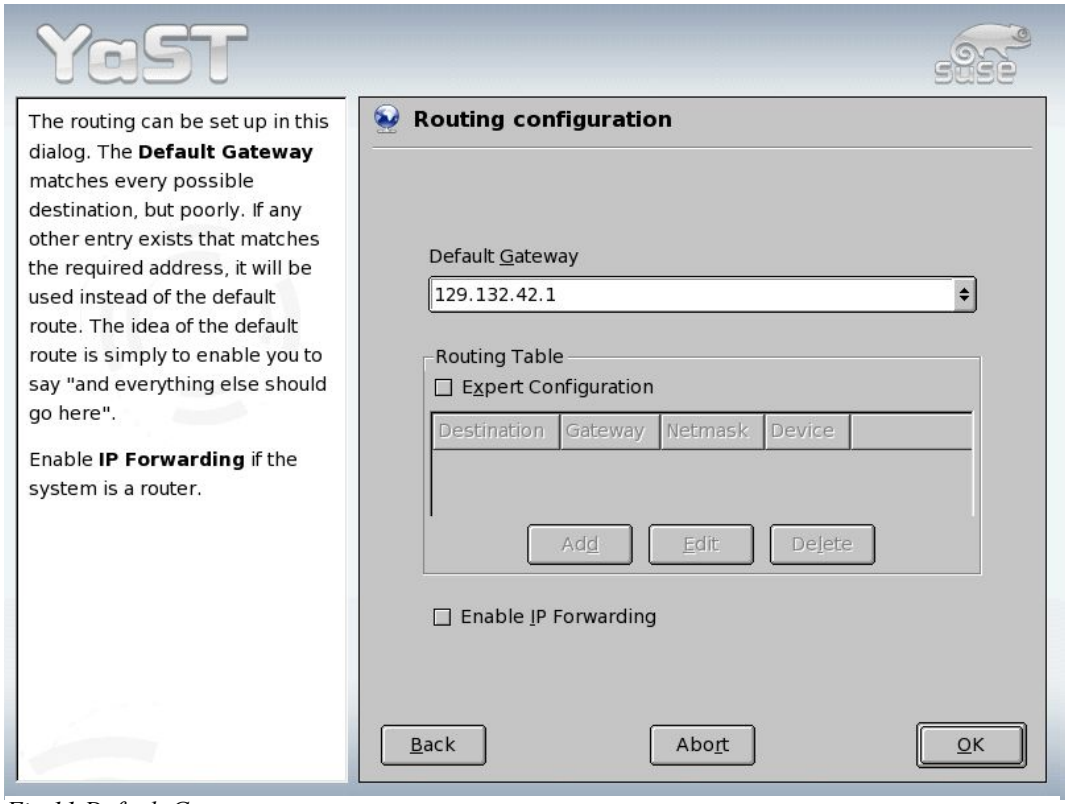


Fig 11 Default Gateway

15. Jetzt lässt sich kurz testen ob eine Verbindung zum Internet möglich ist. Diesen Test sollte man durchführen, nicht aber updates herunterladen (das sollte man erst nach Abschluss der gesamten Installation durchführen, da ein Kernelupdate zum jetzigen Zeitpunkt einige Nebenwirkungen haben kann), (Fig.12)

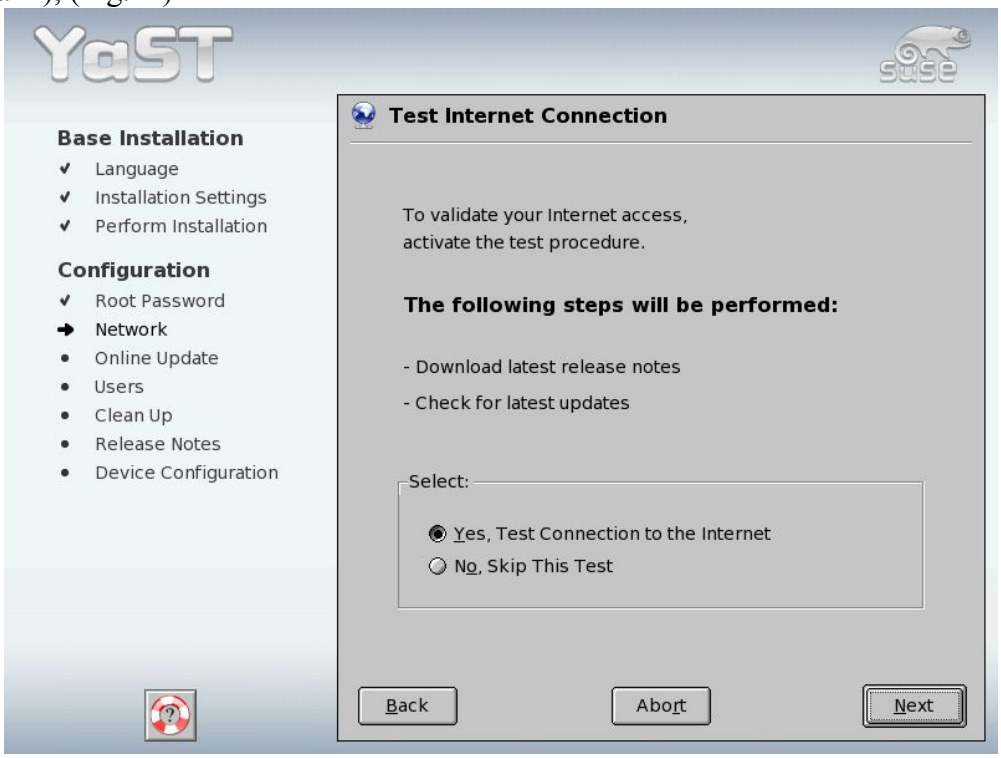


Fig 12 Test der Internetkonnektivität

16. Als nächstes wird gefragt wie die Benutzerauthentifizierung erfolgen soll. Da die Maschine einzelständig funktionieren und keine Benutzer haben soll, wird „Stand-Alone“ ausgewählt (Fig. 13)

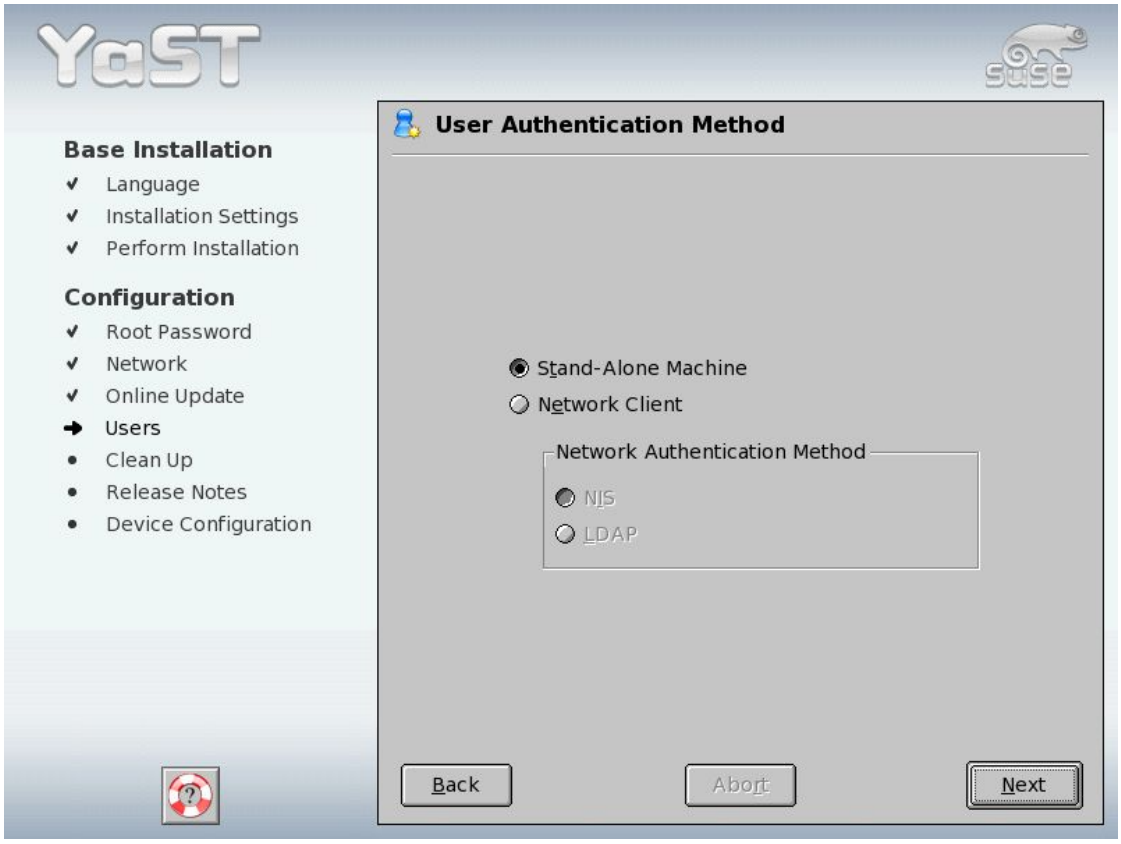


Fig 13 Benutzerauthentifizierung

17. Als nächstes können weitere (non-root) user angelegt werden (Fig. 14). Falls gewünscht können nun Systembenutzer für den Login auf den Server angelegt werden. Dies kann man aber auch überspringen (und dann gegebenfalls direkt als root auf dem Server einloggen)

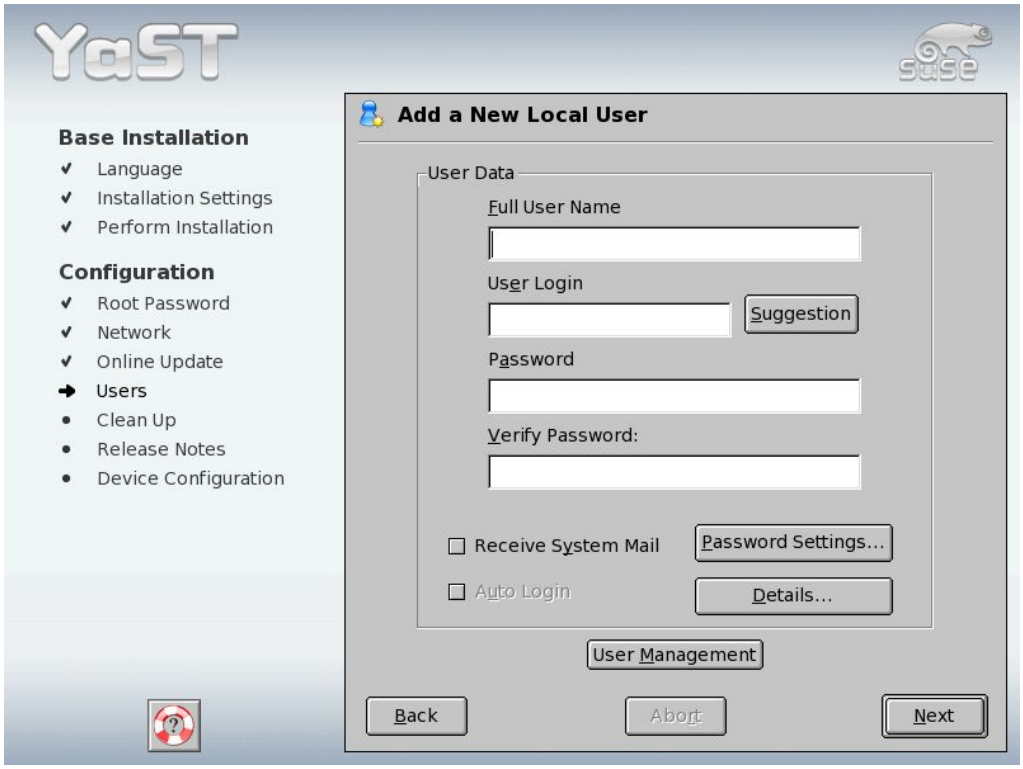


Fig 14 Weitere Benutzer anlegen

18. Schlussendlich schreibt YaST die Systemkonfiguration und führt noch einige Aufräumarbeiten durch. Danach werden die Release-Notes angezeigt und das System bietet einem an, Soundkarten, Drucker und andere Hardware zu konfigurieren, welche für den VPNServer aber irrelevant sind und daher wird dieser Punkt einfach übersprungen.
19. Schliesslich ist die Systeminstallation beendet und wir klicken auf „finish“, was das System komplett aufstartet (Fig. 15)

```
Re-Starting syslog services                                          done
Starting resource manager                                            done
Starting RPC portmap daemon                                          done
Starting mail service (Postfix)                                      done
Starting nfsboot (sm-notify) Backgrounding to notify hosts...
                                                                     done
Starting SSH daemon                                                  done
Starting sound driver                                                done
Starting CRON daemon                                                 done
loading ACPI modules (ac battery button fan processor thermal ) Starting powersa
ved                                                                  done
Loading keymap qwertz/sg-latin1.map.gz                               done
Loading compose table winkeys shiftctrl latin1.add                   done
Start Unicode mode                                                   done
Loading console font lat9w-16.psfu  -m trivial (K                    done
Starting hardware scan on boot                                       done
Starting service xdm                                                 unused
Master Resource Control: runlevel 5 has been                         reached
Skipped services in runlevel 5:                                      xdm


Welcome to SuSE Linux 9.1 (i586) - Kernel 2.6.4-52-default (tty1).


vpnsrv login: _
```

Fig 15 Bootsequenz nach der Installation

20. Die Grundinstallation (von SuSE Linux 9.1) ist damit abgeschlossen
21. Jetzt sollte als erstes als `root` eingeloggt werden und das System durch einspielen von Softwareupdates auf den neuesten Stand gebracht werden. Dazu ruft man (als root) einfach „yast“ auf und wählt im nun Text-gesteuerten Installer das Softwareupdate aus (Fig.16)

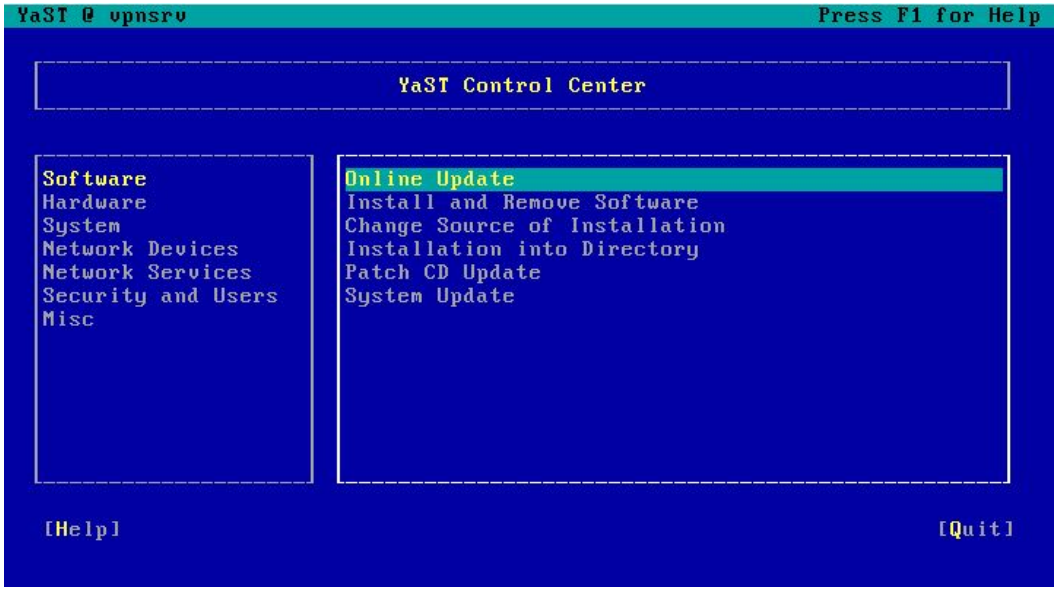


Fig 16 Softwareupdate in yast

22. Im folgenden Dialog bestätigt man die Voreinstellung (oder wählt manuell einen eigenen Server mit den Updatepaketen aus[3]) und beginnt das Softwareupdate.
23. Nach erfolgreichem Update sollte das System durch eingabe von
    `shutdown -r now`
    neu gestartet werden.

3 Empfehlenswert: http://mirror.switch.ch/ftp/pub/suse

Installation der VPNServer Komponenten

1. Zur Sicherheit werden nach dem Reboot die vorangehenden Schritte 21-23 wiederholt, um sicherzustellen, dass alle Updates heruntergeladen wurden.
2. Danach wird die mitgelieferte Datei installer.tar.gz in das Verzeichnis /root kopiert. Dies kann über eine Floppy Disk oder über SCP[4] geschehen.
3. Jetzt wird in dieses Verzeichnis gewechselt und der Installer entpackt:
```
vpnsrv:/ # cd /root
vpnsrv:~ # tar xfvz installer.tar.gz
installer/
installer/modprobe.conf
installer/syslogd.conf
...
```
4. Danach kann in das Installer-Verzeichnis gewechselt und der Installer gestartet werden:
```
vpnsrv:~ # cd installer/
vpnsrv:~/installer # ./vpn-install
```
5. Nun läuft das Installations-Skript durch, installiert die VPN-Serverkomponente, und konfiguriert das System für den Betrieb. Nach Abschluss des Installers wird man aufgefordert, in das Verzeichnis `/opt/vpnsrv/config` zu gehen um die Konfiguration des VPN-Servers abzuschliessen. Mit einem beliebigen Editor wird dazu die Datei `global.conf` editiert.
6. Die wichtigsten Parameter welche unbedingt angepasst werden müssen sind:
    - ADMIN_PASSWORD – Das Administratorpasswort für den VPN-Server
    - INTERNET_IP_ADDRES – Die (öffentliche) IP Adresse, unter welcher der Server erreichbar ist.
    - INTERNET_INTERFACE_NAME – Der Name des Netzwerkinterfaces, auf welchem der Server erreichbar ist.
7. Die Webseiten, welche sich unter /opt/vpnsrv/www befinden, sollten nun auch noch angepasst werden.
8. Danach kann der Server mittels `shutdown -r now` neu gestartet werden und ist ab sofort Einsatzbereit. Ab jetzt kann man z.B. mittels Webbrowser auf die öffentliche IP Adresse auf das VPN-Server Applet verbinden.

4 Ein guter SCP Client für Windows ist WinSCP : http://winscp.sourceforge.net/eng/download.php

Verzeichnisstruktur

Nach erfolgter Installation befinden sich alle Komponenten des VPNServers im Verzeichnis `/opt/vpnsrv`. Die Verzeichnisstruktur dort sieht wie folgt aus:

/opt/vpnsrv

active/	temporäre Verzeichnisse, pro eingeloggtem User ein Verzeichnis.
bin/	Perl- und Shell-skripts um das System zu starten und überwachen sowie die Java-Komponenten von Systemereignissen wie Änderungen an den VPN-Verbindungen zu informieren.
config/	Konfigurationsdateien für das VPNServersystem.
lib/	Libraries für den Javateil des VPNServers.
lib/strategy/	Verzeichnis für die Strategiemodule.
run/	Ablageort von Pid-Dateien. Externe openVPN instanzen legen ihre ProzessIDs hier ab, so dass sie beim shutdown des Systems sauber beendet werden können.
tools/	Toolskripts die während der Entwicklung benutzt wurden
users/	Konfigurationseinstellungen und Accountdaten der Benutzer.
www/	Webseiten, Logdateien und Applet

LogDateien

Zur Laufzeit des VPNServers werden verschiedene Aktivitäten automatisch geloggt und so protokolliert. Die Menge und der Detailgrad dieser Logs kann durch Editieren der `logging.conf` in `/opt/vpnsrv/config` beinflusst werden. Die genaue Syntax der `logging.conf` entspricht der Log4J-Konvention[5].

Hauptsächlich landen die Meldungen in `/var/log/messages`. Ausgaben welche nicht geloggt sondern via System.[out|err].println erzeugt werden, können in `/opt/vpnsrv/nohup.out` eingesehen werden. Schliesslich werden ein- und ausloggende Benutzer in `/var/log/vpnsrv-user.log` separat aufgeführt. Mittels der Zeile:

```
tail -f /var/log/messages /opt/vpnsrv/bin/nohup.out /var/log/vpnsrv-user.log
```

kann der Betrieb „live“ mitverfolgt werden.

Der Detailgrad der Meldungen kann stark schwanken. Auf Level INFO werden relativ wenige jedoch interessante Meldungen geloggt. Auf DEBUG werden hingegen die meisten Nachrichten, welche die Java-Systeme untereinander austauschen notiert werden und die Menge an Log-Ereignissen ist damit sehr hoch. Für den Normalbetrieb sollte der Level nicht über INFO angehoben werden. Kritische Ereignisse werden mit den Levels WARN oder ERROR geloggt; erscheinen also immer wenn sie nicht explizit abgeschaltet werden.

Wichtig: Es ist normal dass eine Exception beim Shutdown geworfen wird. Falls nicht explizit abgeschaltet, erscheint diese auf alle Fälle im Log und ist keine Fehlfunktion, da sie einzig die Reaktion des Systems auf die unerwartete Beendigung des anderen Subsystems reagiert. Detektiert wird dies durch einen missglückten Socket.read() oder write() call, der nach dem einseitigen Abbrechen der TCP-Verbindung auftritt. Die beiden Subsysteme suad und mgmtd senden sich im Abstand von 20 Sekunden dauernd Ping-Pong Pakete zu, um so die Funktionalität und Erreichbarkeit der anderen Seite zu überprüfen. Misslingt dies, was zu besagter Exception führt, so beendet sich die noch laufende Komponente selber. Durch den im Hintergrund laufenden keepalive daemon wird das gesamte System nach spätestens 1 Minute neu gestartet.

Weiterhin kann beim Start des VPNSystems eine Exception von der Log4J Package geworfen werden. Diese Exception reklamiert einen Fehler beim Loggen einer Meldung. Da beim Start des VPNServers auch die Firewall re-initialisiert wird, existiert ein kurzes Zeitfenster, in welchem ein Senden von Nachrichten an den Syslog-Daemon von der Firewall abgelehnt wird, was gelegentlich zu besagter Exception führen kann.

5 http://logging.apache.org/log4j/docs/manual.html

Starten und Stoppen des VPNServers

Das VPNServersystem wird im Hintergrund dauernd über einen in Perl-geschriebenen Mini-Daemon namens `keepalived.pl` überwacht. Dieser Daemon sendet alle 20 Sekunden eine Nachricht an den Suad um dessen Lebendigkeit zu überprüfen. Sollte der Test dreimal versagen, so startet der keepalived den Suad neu. Dieser unter vollen root-Rechten laufende Java-Daemon wiederum re-initialisert die Firewall und started den Mgmtd unter dem nicht-privilegierten Benutzer „vpnsrv“.

Wegen dieser Tatsachen kann der VPNServer nicht einfach beendet werden ohne dass er nicht durch den `keepalived.pl` gleich wieder gestartet wird. Wenn man das System ohne herunterfahren des Rechners beenden will, so muss via

```
/etc/init.d/vpnsrv stop
```

der keepalived.pl zuerst beendet werden. Mittels

```
/etc/init.d/vpnsrv start
```

kann das VPNServersystem später wieder gestartet werden. Der Betrieb des Apache2-Webservers wird damit aber nicht beeinflusst. Sollte der Webserver beendet werden, so kann dies über

```
rcapache2 stop
```

erreicht werden. Sowohl der Web- wie auch der VPNServer werden automatisch beim bootvorgang des Rechners gestartet.

Schlussbemerkungen und Hinweise für eine Weiterentwicklung

Folgend noch eine kurze Auflistung von Punkten, welche man im Zuge einer eventuellen Weiterentwicklung dieser Semesterarbeit noch verbessern, respektive erweitern könnte:

- Das Applet ist funktionell, aber weder schön noch intuitiv zu bedienen. Insbesondere die Darstellung der Netzwerktopologie ist nicht gelungen. Hier könnte man noch etwas verbessern
- Das Applet kennt nur bi-direktionale Verbindungen, obwohl das restliche System eigentlich mit uni-direktionalen Verbindungen zwischen zwei Knoten arbeitet.
- Das jetzige System kennt nur ein virtuelles Netzwerk zur gleichen Zeit. Prinzipiell könnte man aber über einen VPNServer mehrere Simulationen und damit Netzwerke parallel betreiben.
- Im Augenblick gibt es eine hart-kodierte Limite von maximal 253 Knoten, durch die Tatsache dass jedem Knoten/Benutzer eine IP Adresse automatisch zugeteilt werden muss und keine automatische Allokation über die Grösse eines Klasse-C Netzwerkes hinaus programmiert wurde.
- Es findet keine genaue Überprüfung der über die Sockets empfangenen Daten für die Kommunikation zwischen Suad und Mgmtd sowie Mgmtd und dem Applet statt.
- Das System könnte relativ einfach durch QueueDisciplines[6] wie HTB oder Netem[7] zusätzlich per link bandwith shaping machen oder delay simulieren. Weiterhin könnte über die optionalen Netfilter Matches Nth[8] und Random[9] jedes X-te Packet oder durch eine Wahrscheinlichkeit gesteuert Pakete verworfen werden, womit man Packetloss simulieren kann.

6 http://linux-ip.net/articles/Traffic-Control-HOWTO/components.html#c-qdisc
7 http://developer.osdl.org/shemminger/netem/
8 http://netfilter.org/patch-o-matic/pom-base.html#pom-base-nth
9 http://netfilter.org/patch-o-matic/pom-base.html#pom-base-random

Anhang 1: Debug „Cheat“-codes

Der Suad reagiert auf seinem lokalen NOTIFY_PORT[10] auf spezielle Befehle. Mittels ssh auf dem VPNServer eingeloggt, kann mittels

```
telnet localhost <NOTIFY_PORT>
```

mit dem Suad kommuniziert werden. Folgende Befehle werden erkannt:

- „QUIT“: Beendet sofort den Suad.
- „UP <tapNr>“: Simuliert das Verbinden eines openVPN Benutzers über <tapNr>. Beispiel: `UP tap12`
- „DOWN <tapNr>“:Simuliert das Trennen des openVPNBenutzers von <tapNr>. Beispiel: `DOWN tap12`
- „ALLOW <tap1>:<tap2>“: Erlaubt die Kommunikation zwischen den beiden Benutzern der angegebenen Tap-Nummern. Beispiel: `ALLOW tap12:tap14`
- „DISALLOW <tap1>:<tap2>“: Verbietet die Kommunikation zwischen den beiden Benutztern der angegebenen Tap-Nummern. Beispiel: `DISALLOW tap12:tap14`
- „DEBUG“: Schreibt das aktuelle Netzwerklayout über stdout aus (-> `/var/log/messages`)

10 Defaultmässig 713

Anhang 2: Spezielles

IP-Wechsel

Da openVPN auf seine lokale IP-Adresse konfiguriert werden muss, steht die in `INTERNET_IP_ADDRESS` angegebene IP Adresse in sämtlichen openVPN Konfigurationsdateien in `/opt/vpnsrv/users`. Im Falle eines Wechsels der IP Adresse nachdem bereits Benutzter angelegt wurden, muss nicht nur die Konfigurationsdatei in global.conf angepasst werden, sondern auch alle Benutzerkonfigurationsdateien (2 pro Benutzer, eine für das lokale Ende („.conf“) und einmal die Konfigurationsdateien welche die Benutzer jederzeit über das Applet neu anzeigen lassen können („.ovpn“)). Weiterhin müssen die bereits registrierten Benutzer natürlich ihre lokale Konfiguration ebenfalls anpassen, damit sie den nun unter neuer IP Adresse funktionierenden Server weiterhin erreichen zu können.

Anhang 3: Wie schreibt man ein Strategie-Modul

Ein Strategie-Module ist eine simple Java-Klasse, welche das Interface „NetworkBuilder“ implementiert. Durch die von diesem Interface definierten Methoden enthält das Strategie-Modul die Informationen wann es starten oder stoppen soll, sowie die Änderungen an der Anzahl von Netzwerkknoten mitgeteilt. Auf solche Ereignisse hin aktiviert, kann es über das „NetMap“API Verbindungen zwischen diesen Knoten aufbauen (zulassen) oder abbauen (resp. verbieten) um so eine Netzwerktopolgie zu schaffen.

Danach reicht es, die Klassendatei eines Strategie-Moduls in das dafür vorgesehene Verzeichnis zu kopieren, damit es ab dem nächsten Start des Systems als gültiges Modul ausgewählt und konfiguriert werden kann. Standardmässig ist dies `/opt/vpnsrv/lib/strategy`.

Methoden von NetworkBuilder

Folgende Methoden sind durch das Interface NetworkBuilder vorgegeben und müssen von jedem Strategie-Modul implementiert werden:

- public String getName (): Hierüber soll sich das Strategie-Modul mit einem möglichst aussagekräftigen Namen identifiezieren.
- public void init (NetMap map): Diese Methode wird vom Mgmtd aufgerufen, damit sich das Strategie-Modul vorbereiten kann. Die übergebene Referenz auf ein Objekt des Typs NetMap ist sehr wichtig und muss unbedingt im Modul gespeichert werden. Über NetMap wird die Netzwerktopologie gesteuert und Informationen über die Knoten und Verbindungen abgerufen.
- public void start (): Diese Methode wird vom Mgmtd aufgerufen, sobald das Strategie-Modul beginnen soll, die Topologie zu kontrollieren. Ab diesem Zeitpunkt und bis stop() aufgerufen wird, ist dieses Modul für die Topologie verantwortlich. Sollte das Modul komplexere Operationen durchführen, so sollte es jetzt einen eigenen Thread dafür starten.
- public void stop (): Diese Methode wird vom Mgmtd aufgerufen, sobald ein anderes Modul die Zuständigkeit übernimmt. Ab sofort darf dieses Modul keinerlei Änderungen mehr an der Topologie vornehmen, bis ein weiteres Mal start() aufgerufen wurde.
- public void nodeAdded (Node n): Diese Methode wird vom Mgmtd aufgerufen, sobald ein neuer Knoten im System hinzugefügt wurde – oder genauer – sobald eine neue VPN-Verbindung eines Knotens registriert wurde. Durch die übergebene Referenz muss der neue Knoten nicht umständlich über NetMap ausfindig gemacht werden. Jetzt sollte das Modul entsprechend seiner Vorstellung die Topologie ändern oder mit dem neuen Knoten als Teil des Netzwerkes neu aufbauen.
- public void nodeRemoved (Node n): Diese Methode wird aufgerufen, sobald ein Knoten sich aus dem Netzwerk entfernt hat, also seine VPN-Verbindung zum Server nicht mehr besteht. Das Strategie-Modul muss die Verbindungen die von diesem Knoten ausgingen oder zu ihm führten nicht entfernen, da dies automatisch geschieht. Jedoch

muss eventuell nun die Topologie, je nach Aufgabe des Strategie-Modules, geändert werden um Ersatzverbindungen herzustellen.

- public void externalStructureChange (): Wird aufgerufen, sobald eine „externe Änderung“ an der Topolgie des Netzwerkes vorgenommen wurde. Dies passiert genau dann, wenn ein Administrator über das Applet manuell eingreift und so Verbindungen entfernt oder hinzufügt. Durch diese Methode kann eine Strategie eingreifen und „ihre“ Topologie sofort wieder reparieren und damit dem Administrator entgegenwirken – oder aber sie nimmt einfach zur Kenntnis, dass etwas geändert hat. Auf alle Fälle sollte ein Strategie-Modul dieses Ereignis zur Kenntnis nehmen und eventuell vorhandene interne Annahmen gegebenfalls überprüfen und anpassen.

An dieser Stelle wird auch auf die vorhandene, etwas ausführlichere Javadoc-Dokumentation der Klasse NetworkBuilder verwiesen, deren Hinweise beim Schreiben eines Strategie-Modules auf alle Fälle gelesen werden sollten

Das NetMap API

Über die an die NetworkBuilder.init() Methode übergebene Referenz auf das NetMap-Objekt steht die eigentliche API zur Manipulation der Verbindungen und der Abfrage der momentanen Topologie zur Verfügung. Es folgt eine Auflistung der wichtigsten Methoden von NetMap für die Funktion eines Strategie-Modules. Die komplette Dokumentation befindet sich auch hier im Javadoc-Format in NetMap selber. Viele Methoden dienen der internen Kommunikation zwischen den zwei Java-subsystemen und werden deshalb für ein Strategie-Modul nicht benötigt:

- public int getNumNodes (): Gibt die Anzahl der Knoten zurück.
- public int getNumConnections (): Gibt die Anzahl der Kanten/Verbindungen zurück. Es ist wichtig zu Wissen, dass Verbindungen grundsätzlich unidirektional sind. Um eine bi-direktionale Verbindung zwischen zwei Knoten herzustellen, müssen also 2 Verbindungen erstellt werden. Das Applet erstellt oder entfernt immer zwei Verbindungen, wenn man manuell im Graph Verbindungen zeichnet.
- public Node getNodeByDevice (String device): Jeder Knoten trägt einen „Device-Namen“. Dieser entspricht immer dem Tap-Devicenamen auf dem Serverssystem, also z.B. `tap12`.
- public Connection getConnectionByName (String name): Jede Verbindung kann durch ihren Namen beschrieben werden. Dieser besteht aus den Device-Namen der zwei Knoten getrennt durch einen Doppelpunkt, also z.B. `tap12:tap22`.
- public void addConnection (Node from, Node to): Fügt eine neue Verbindung zwischen den beiden angegebenen Knoten hinzu, mit dem ersten Knoten als Start (Sender) und dem zweiten Knoten als Ziel (Empfänger).
- public void removeConnection (Connection c): Entfernt die übergebene Verbindung.
- public void removeConnection (Node from, Node to): Entfernt eine eventuell vorhandene

Verbindung ausgehend vom Knoten `from` zum Knoten `to`.

- public ArrayList getNodes (): Erzeugt und übergibt eine Liste aller Knotenelemente. Da diese Liste eine Kopie ist, darf sie beliebig manipuliert werden.
- public ArrayList getConnections (): Erzeugt und übergibt eine Liste aller Verbindungen. Da diese Liste eine Kopie ist, darf sie beliebig manipuliert werden.

Beispielimplementation eines Strategie-Moduls

Folgend eine Beispielimplementation eines primitiven Strategie-Modules, welches neue Knoten linear mit dem jeweils letzten hinzugekommenen Knoten verbindet. Es reagiert nicht auf externe Veränderungen, passt die Topologie jeweils nur an wenn ein Konten hinzugefügt wird und repariert nichts wenn es später bei bereits bestehender zufälliger Topologie (wieder) aktiviert wird. Dieses Strategie-Modul steht unter dem Namen „DummyLineBuilder“ in der Default-Installation zur Verfügung:

```
/*
 * Created on 09.08.2004
 *
 */
package ch.ethz.dcg.vpnsrv.data.strategy;
import ch.ethz.dcg.vpnsrv.data.NetMap;
import ch.ethz.dcg.vpnsrv.data.NetworkBuilder;
import ch.ethz.dcg.vpnsrv.data.Node;
/**
 * @author Drax
 *
 * Simple dummy implementation of a NetwerkBuilder strategy. It just linearly
 * concatenates the latest two nodes. If a node is removed, it fixes the
 * hole.<br>
 * Does not have an independent thread and does ignore all external changes.
 */
public class LinearStrategy
        implements NetworkBuilder
{
        //---- Static
        public final static String name = "DummyLineBuilder";
        //---- Fields
        private Node last;
        private NetMap map;
        private boolean running;

        //---- Constructor
```

```
public LinearStrategy () {
	super();
	running = false;
}
//---- Methods
//---- NetworkBuilder
/**
 * @see ch.ethz.dcg.vpnsrv.data.NetworkBuilder#getName()
 */
public String getName() {
	return LinearStrategy.name;
}
/**
 * @see ch.ethz.dcg.vpnsrv.data.NetworkBuilder#init(ch.ethz.dcg.vpnsrv.data.NetMap)
 */
public void init(NetMap map) {
	this.map = map;
	last = null;
}
/**
 * @see ch.ethz.dcg.vpnsrv.data.NetworkBuilder#start()
 */
public void start() {
	running = true;
}
/**
 * @see ch.ethz.dcg.vpnsrv.data.NetworkBuilder#stop()
 */
public void stop() {
	running = false;
}
/**
 * @see ch.ethz.dcg.vpnsrv.data.NetworkBuilder#nodeAdded(ch.ethz.dcg.vpnsrv.data.Node)
 */
public void nodeAdded(Node n) {
	if(!running)
		return;
	if(last == null) { // no node seen yet, do nothing.
		last = n;
	} else { // connect this to last and reverse direction. update last
```

```
                map.addConnection(last, n);
                map.addConnection(n, last);
                last = n;
            }
        }
        /**
         * @see ch.ethz.dcg.vpnsrv.data.NetworkBuilder#nodeRemoved(ch.ethz.dcg.vpnsrv.data.Node)
         */
        public void nodeRemoved(Node n) {
            if(!running)
                return;
            // do nothing except if "last" node removed
            if(last != null &&
                last == n) {
                // need new last
                if(map.getNumNodes() > 0) {
                    last = (Node)map.getNodes().get(0);
                } else {
                    last = null;
                }
            }
        }
        /**
         * @see ch.ethz.dcg.vpnsrv.data.NetworkBuilder#externalStructureChange()
         */
        public void externalStructureChange() {
            // completly and utterly ignored
        }
}
```

Weitere, auch komplexere Strategie-Module finden sich im Source-Code unter dem Package „ch.ethz.dcg.vpnsrv.data.strategy“ wieder.

Anhang 4: Verwendete fremde Software

Die folgende Liste an Software ist Bestandteil des VPNServer-Projektes und nicht im Zuge der Semesterarbeit entwickelt worden:

- Der Linux Kernel http://www.linux.org
- SuSE Linux Distribution http://www.suse.de
- Bridge-Filter und ebtables http://ebtables.sourceforge.net/
- openVPN http://openvpn.sourceforge.net/
- IPTables + Netfilter Firewall http://www.netfilter.org/
- Apache2 Webserver http://www.apache.org
- Log4J http://logging.apache.org/log4j/docs/

Weiterhin wurde die grafische Darstellung des Netzwerklayoutes im Applet aus dem Sun JAVA SDK unter beachtung der geltenden Lizenzbedingung entliehen. Das Original befindet sich unter j2sdk1.4.2/demo/applets/GraphLayout/example4.html.

Die restliche Software, insbesondere die in Java geschriebenen Hauptbestandteile Suad, Mgmtd, VPNServer-Applet sowie die notwendigen Perl- und Bash-Skripte zur Steuerung der Subsysteme untereinander sind hingegen während der Semesterarbeit entwickelt worden.

Anhang 5: Manuelle Paketauswahl

Sollte die Verwendung einer Diskette oder ähnlichem für die Paketselektion zur Installation nicht zur Verfügung stehen, so kann die Auswahl auch manuell erfolgen, was allerdings ein gewisser zeitlicher Aufwand bedeutet. Die empfohlene Vorgensweise hierbei ist, die Auswahl „minimales grafisches System“ anzuwählen und danach über den Manuel-Knopf in der detailierten Paketauswahl sicherstellen, dass folgende Pakete installiert werden:

```
3ddiag-0.716-108
CheckHardware-0.1-952
RealPlayer-8.0.3.465-89
SuSEfirewall2-3.1-310.3
WindowMaker-0.80.2.20030506-197.2
WindowMaker-applets-1.0-642
WindowMaker-themes-0.1-238
XFree86-4.3.99.902-43.28
XFree86-Mesa-4.3.99.902-40
XFree86-Xvnc-4.3.99.902-43.28
XFree86-fonts-75dpi-4.3.99.902-40
XFree86-fonts-scalable-4.3.99.902-40
XFree86-libs-4.3.99.902-43.31
XFree86-server-4.3.99.902-43.28
XFree86-server-glx-4.3.99.902-40
aaa_base-9.1-0.26
aaa_skel-2003.9.18-93
acl-2.2.21-54.4
acroread-5.09-4.2
alsa-1.0.3-37
apache2-2.0.49-27.16
apache2-doc-2.0.49-27.16
apache2-prefork-2.0.49-27.16
ash-0.4.18-56
at-3.1.8-894
attr-2.4.12-56
autoyast2-2.9.34-1
autoyast2-installation-2.9.34-1
bash-2.05b-305.1
bc-1.06-744
bind-utils-9.2.3-76
binutils-2.15.90.0.1.1-31
bitstream-vera-1.10-163
bridge-utils-0.9.6-121
busybox-1.00.pre8-26.3
bzip2-1.0.2-344
cabextract-1.0-15
convmv-1.07-53
coreutils-5.2.1-23
cpio-2.5-308
cpp-3.3.3-41
cracklib-2.7-1006
cron-3.0.1-920
curl-7.11.0-39
cyrus-sasl-2.1.18-29
db-4.2.52-85
desktop-data-SuSE-9.1-8
device-mapper-1.00.09-12
devs-9.1-0
dhcpcd-1.3.22pl4-193
dialog-0.9b-188
diffutils-2.8.4-75
e2fsprogs-1.34-115
ed-0.2-864
eject-2.0.13-185
```

```
ethtool-1.8-123
evms-2.3.1-7
expat-1.95.7-37
fbset-2.1-778
file-4.07-48
filesystem-9.1-0
fillup-1.42-98
findutils-4.1.7-860
finger-1.2-39
fontconfig-2.2.92.20040221-28
freeglut-2.2.0-78
freetype2-2.1.7-53
fribidi-0.10.4-481
fvwm2-2.5.9-38
gawk-3.1.3-205
gcc-3.3.3-41
gdbm-1.8.3-225
ghostscript-fonts-std-7.07.1rc1-190
glib-1.2.10-586
glib2-2.2.3-117
glibc-2.3.3-97
glibc-devel-2.3.3-97
glibc-locale-2.3.3-98
gnome-filesystem-0.1-172
gpart-0.1h-475
gpg-1.2.4-68.4
gpm-1.20.1-299
grep-2.5.1-416
groff-1.17.2-876
grub-0.94-25
gtk-1.2.10-877
gzip-1.3.5-136
hdparm-5.5-41
heimdal-lib-0.6.1rc3-51
hermes-1.3.2-443
hotplug-0.44-32.22
hwinfo-8.62-0.2
ifnteuro-1.2.1-187
imlib-1.9.14-180.8
imwheel-0.9.5-1027
info-4.6-61
initviocons-0.4-288
insserv-1.00.2-85
intlfnts-1.2.1-187
iproute2-2.4.7-861
iptables-1.2.9-95.5
iputils-ss021109-147
isapnp-1.26-489
java2-jre-1.4.2-129
jfsutils-1.1.5-20
kbd-1.12-26
kernel-default-2.6.5-7.108
kernel-source-2.6.5-7.108
ksymoops-2.4.9-135
ldapcpplib-0.0.3-16
less-382-34.7
libacl-2.2.21-54.4
libapr0-2.0.49-27.16
libattr-2.4.12-56
libcap-1.92-479
libgcc-3.3.3-41
libjpeg-6.2.0-731
liblcms-1.12-55
libmng-1.0.6-54
libpcap-0.8.1-39
libpng-1.2.5-182.10
```

```
libselinux-1.8-16
libstdc++-3.3.3-41
libstroke-0.4-733
libtiff-3.6.1-38.3
libungif-4.1.0b1-581
libusb-0.1.8-31
libxcrypt-2.1.90-61
libxml2-2.6.7-28
libxslt-1.1.2-58
liby2util-2.9.25-0.3
lilo-22.3.4-357
logrotate-3.7-31
lsof-4.70-30
lukemftp-1.5-578
mailx-10.6-61
make-3.80-184
man-2.4.1-209
mc-4.6.0-324.7
mdadm-1.5.0-40
mingetty-0.9.6s-73
mkinitrd-1.0-199.50
mktemp-1.5-729
module-init-tools-3.0_pre10-37.5
ncurses-5.4-61.3
net-tools-1.60-543
netcat-1.10-864
netcfg-9.1-0.3
ntfsprogs-1.9.0-21
openldap2-client-2.2.6-34
openslp-1.1.5-73
openslp-server-1.1.5-73
openssh-3.8p1-33
openssh-askpass-3.8p1-33
openssl-0.9.7d-15.13
openvpn-1.5.0-46
pam-0.77-221
pam-modules-9.1-0.7
parted-1.6.6-138
patch-2.5.9-141
pciutils-2.1.11-192
pcre-4.4-109
perl-5.8.3-32
perl-Config-Crontab-1.03-46
perl-Digest-SHA1-2.07-30
perl-XML-Parser-2.34-28
perl-gettext-1.01-576
permissions-2004.7.30-0.2
popt-1.7-176.3
portmap-5beta-728
postfix-2.0.19_20040312-11
powersave-0.7-7
procinfo-18-35
procmail-3.22-35
procps-3.2.1-4
providers-2004.4.2-4
psmisc-21.4-39
pwdutils-2.6.4-2.1
qt3-3.3.1-36.16
raidtools-1.00.3-222.3
readline-4.3-306
recode-3.6-488
reiserfs-3.6.13-24
release-notes-9.1-4
resmgr-0.9.8-47
rpm-4.1.1-177
rsh-0.17-548
```

```
sash-3.7-28
sax2-4.8-98
saxident-1.1-1131
saxtools-2.2-1426
scpm-0.9.6-34
scsi-1.7_2.34_1.06_0.11-5
sed-4.0.9-31
siga-9.100-29
sitar-0.8.11-17
slang-1.4.9-121
src_vipa-2.0.0-59
submount-0.9-33.6
suse-build-key-1.0-658
suse-release-9.1-0
sysconfig-0.31.0-15.8
syslogd-1.4.1-519
sysvinit-2.85-20
tar-1.13.25-298
tcl-8.4.6-26
tcpd-7.6-710
tcpdump-3.8.1-49
tcsh-6.12.00-448
telnet-1.1-38
terminfo-5.4-59
tightvnc-1.2.9-177
timezone-2.3.3-98
tk-8.4.6-37
udev-021-36
unclutter-8-830
unixODBC-2.2.8-55
usbutils-0.11-211
utempter-0.5.2-385.3
util-linux-2.12-72.15
vim-6.2-233
w3m-0.4.1_m17n_20030308-198
wget-1.9.1-45
xanim-2.80.2-764
xaw3d-1.5E-212
xbanner-1.31-854
xdg-menu-0.2-45
xdmbgrd-0.5-26
xf86tools-0.1-949
xfsprogs-2.6.3-29
xlockmore-5.11.1-81
xntp-4.2.0a-23
xtermset-0.5.2-118
yast2-2.9.60-5
yast2-backup-2.9.16-9
yast2-bootloader-2.9.22-1
yast2-control-center-2.9.11-3
yast2-core-2.9.94-1.3
yast2-country-2.9.19-9
yast2-dhcp-server-2.9.16-5
yast2-dns-server-2.9.15-14
yast2-firewall-2.9.11-14
yast2-inetd-2.9.12-12
yast2-installation-2.9.57-1
yast2-kerberos-client-2.9.8-14
yast2-ldap-2.9.10-10
yast2-ldap-client-2.9.14-5
yast2-mail-2.9.13-14
yast2-mail-aliases-2.9.13-14
yast2-mouse-2.9.10-10
yast2-ncurses-2.9.26-0.3
yast2-network-2.9.57-0.2
yast2-nfs-client-2.9.11-14
```

```
yast2-nfs-server-2.9.9-14
yast2-nis-client-2.9.15-11
yast2-nis-server-2.9.9-0.2
yast2-ntp-client-2.9.10-14
yast2-online-update-2.9.11-3
yast2-packagemanager-2.9.52-0.2
yast2-packager-2.9.43-3
yast2-pam-2.9.13-0.2
yast2-perl-bindings-2.9.25-8
yast2-phone-services-2.9.5-12
yast2-power-management-2.9.8-14
yast2-powertweak-2.9.14-10
yast2-profile-manager-2.9.7-11
yast2-qt-2.9.24-0.4
yast2-repair-2.9.10-14
yast2-restore-2.9.10-13
yast2-runlevel-2.9.13-14
yast2-scanner-2.9.16-0.2
yast2-security-2.9.14-9
yast2-slp-2.9.9-2
yast2-sound-2.9.19-4
yast2-storage-2.9.37-0
yast2-support-2.9.3-14
yast2-sysconfig-2.9.14-10
yast2-tftp-server-2.9.4-14
yast2-theme-SuSELinux-2.9.9-6
yast2-trans-en_US-2.9.5-5
yast2-transfer-2.9.2-10
yast2-tune-2.9.20-10
yast2-tv-2.9.9-6
yast2-update-2.9.23-2
yast2-users-2.9.29.1-0.1
yast2-x11-2.9.9-10
yast2-xml-2.9.8-10
zlib-1.2.1-70.6
zsh-4.2.0-31
```