Universität Koblenz-Landau

Fachbereich 4 (Informatik)
Institut für Informatik
Arbeitsgruppe Rechnernetze

Verteilte Simulationen und externe Verbindungen mit Virtual-Network User-Mode Linux

Studienarbeit

Keupen, Tim (202110012)
Betreuer: Prof. Dr. Ch. Steigner, Dipl.Inf. H. Dickel
Koblenz, 13. März 2006

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig, ohne fremde Hilfe und ohne Benutzung anderer als den von mir angegebenen Quellen angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet. Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Koblenz, den 13. März 2006

Inhaltsverzeichnis

1 Einleitung

1.1 Vorraussetzungen

Alle Beispiele dieser Arbeit beziehen sich auf die VNUML Programm-Version 1.5. Mit Hostrechner bzw. Hostsystem sei im folgenden der physikalisch vorhandene Rechner gemeint, auf dem die Simulation abläuft. Die Instanzen der UML Prozesse sind mit virtueller Maschine (VM) bzw. virtueller Rechner bezeichnet.

1.2 Motivation

Die Idee virtuelle Maschinen zu verwenden ist nicht neu. Schon in den 70er Jahren konnten Mainframes von IBM sogenannte Gastbetriebssysteme starten. Ein klassisches Beispiel ist das IBM VM/370. Dieses System gab jedem Benutzer den Eindruck, er würde direkt und exklusiv die Hardware (Prozessor, Hauptspeicher etc) verwenden. [8]

Der Trend hin zu Einzelplatzrechnern (Desktop-PCs) ließ solche Konzepte einschlafen. In den letzten Jahren wurden die Ideen der 70er jedoch wieder neu belebt. Die anhaltend steigende Leistung der heute verfügbaren Hardware ermöglicht es, Multisysteme auf einem einzigen Gerät simultan zu betreiben. Beim sogenannten Server-Hosting läuft auf einer physikalischen Maschine für jeden Kunden eine eigene virtuelle Maschine, die die gemieteten Dienste (z.B. Webserver oder Mailserver) anbietet. Untereinander dürfen sich diese virtuellen Maschinen nicht beeinflussen können. User-Mode Linux bietet hier auf Hardwareebene die Möglichkeit die Adressräume der Prozesse zu kapseln und ist somit für Systeme, die nicht im High-Perfomance Bereich angesiedelt sind, durchaus interessant.[5]

Da in einer Netzwerksimulation mit VNUML jeder Netzknoten durch eine virtuelle Maschine repräsentiert wird, deren komplette Hardware vom Host emuliert wird, steigt der Anspruch an die Performanz des Hostsystems mit jeder zusätzlichen VM. Ab einem gewissen Grad ist ein einzelner Rechner nicht mehr in der Lage ein Szenario zu starten, bzw. Topologieuntersuchungen darauf durchzuführen. Die Lösung liegt hier in der verteilten Simulation, bei der ein Szenario auf mehreren verschiedenen Hostsystemen ausgeführt wird, wobei jeder Host einen bestimmten Teil übernimmt. Die Netztopologie ändert sich dabei für den Benutzer nicht (Beispiel: Ein `traceroute` von VM A zu VM B erscheint immer identisch, dabei soll es unerheblich sein, ob A und B innerhalb eines Host oder auf zwei verschiedenen Host simuliert werden.). In dieser Arbeit werden „verteilte Szenarios“ ebenso untersucht, wie die direkt damit zusammenhängende Möglichkeit, Netze eines virtuellen Szenarios mit echten physikalischen Netzen (z.B. dem Internet) zu verbinden.

2 Grundlagen

In diesem Kapitel soll ein Überblick über User-Mode-Linux und die darauf aufbauende Funktionsweise des Netzwerksimulators VNUML geschaffen werden.

2.1 User-Mode Linux

„User-Mode Linux“ (UML) wird seit 1999 von Jeff Dike als Open-Source Projekt entwickelt. Ursprünglich wurde das Projekt initiiert um das Debuggen des Linux-Kernels zu vereinfachen. Verwendet wird hierzu ein Linux-Kernel, der nicht direkt auf die Hardware aufsetzt, sondern als Prozess in einem Hostsystem läuft. Wie jede Anwendung wird ein UML-Kernel bei Bedarf gestartet und kann ebenso leicht auch wieder beendet werden. Dabei läuft dieser Kernel im sogenannten „Userspace“, hat also nicht die volle Gewalt über das System.

UML erweitert das bekannte Schichtenmodell eines Betriebssystems (Hardware, Kern, Prozesse) um eine weitere Schicht. Der UML-Kernel läuft auf Ebene der Prozesse über dem echten Kernel, erst darüber laufen die innerhalb der UML-Instanz nicht-privilegierten Anwendungen im Userspace. UML-Kernel und Anwendungen greifen also nicht direkt auf die Hardware zu, Systemaufrufe werden an den Hostkernel weitergereicht.

Mit UML können mehrere virtuelle Linux-Systeme („Gäste“) gleichzeitig auf demselben physischen Linux-System („Host“) ausgeführt werden. Jeder Gast kann dabei unterschiedliche Versionen des Linux-Kernels (z.B. 2.4.28, 2.6.9-bb4) ausführen und auf verschiedenen Linux-Distributionen (z.B. SuSE Professional 9.3, Debian 3.0, Redhat,..) basieren.[5]

2.2 VNUML

„Virtual-Network User-Mode Linux“ (VNUML) ist ein Tool zur Simulation von Computernetzen. Durch die Verwendung von User-Mode Linux (UML) können virtuelle Rechner in einem Szenario erzeugt werden, die nahezu alle Eigenschaften eines realen Linux-Rechners haben. Dabei wird jeder virtuelle Rechner als ein Prozess (mit eigenem Filesystem etc) auf einem Rechner ausgeführt. Das ermöglicht entsprechend der Performanz des Hostrechners den Aufbau und Test fast beliebiger Netzwerktopologien, ohne die dazu erforderliche Hardware selbst zu besitzen. Die virtuellen Maschinen können beliebig konfiguriert werden und in einer Simulation verschiedene Rollen einnehmen (z.B. Router, Webserver, Client, etc).

Da es sich bei den virtuellen Rechnern um Prozesse mit einem eigenen Kernel, Dateisystem und Arbeitsspeicher handelt, ist es möglich, nahezu beliebige Programme (ohne besondere Hardwareanforderungen) auf ihnen zu installieren, die dann im Laufe der Simulation ausgeführt

werden können. Welche Simulation ausgeführt wird (Beispiel: jede VM nimmt die Rolle eines BGP-Routers ein), wird definiert durch eine Folge von Kommandos, die auf jedem virtuellen Rechner beim Aufruf einer Startsequenz ausgeführt werden. Somit eignet sich VNUML z.B. zum Testen von verteilter Software oder zur Untersuchung von Netzwerkprotokollen. VNUML besteht aus zwei Komponenten:

VNUML-Sprache Die XML-Sprache von VNUML dient dem Beschreiben von Szenarios. Eine XML-Datei beschreibt ein Szenario.

VNUML-Parser Der VNUML-Parser baut die Netzwerktopologie entsprechend der XML-Beschreibung mit Hilfe von User-Mode Linux auf. Die komplexen Details von UML werden vor dem Nutzer versteckt.

Die virtuellen Netze zwischen den virtuellen Maschinen werden dabei nicht simuliert (Paketaustausch findet auf Ebene der Interprozesskommunikation statt). Performance-Messungen einer Netzwerktopologie durch Erzeugung von hohem Datenverkehr sind nicht möglich, da die Leitungen innerhalb der Netze nicht modelliert werden.[1]

2.3 Schnittstellen im Hostsystem

Während dem Start eines Szenarios werden vom Parser innerhalb des Hostsystems mehrere (virtuelle) Schnittstellen angelegt. Diese sind der essentielle Teil der Netzwerksimulation, da sich damit die virtuellen Maschinen in Netzen verbinden und vom Hostsystem aus konfigurieren lassen. Innerhalb des Host lassen sich nach Aufbau der Topologie insgesamt vier Arten virtueller Schnittstellen unterscheiden:

Host Interfaces sind die hardwareabhängig, bzw. durch die Linux- Konfiguration bestimmten, verfügbaren Schnittstellen innerhalb des Host. Diese werden **nur** dann verändert, wenn ein Szenario Verbindungen zu externen Netzen erhalten soll. Dabei verliert die entsprechende Schnittstelle ihre IP-Adresse und wird in den „promisc“-Mode gesetzt, um auf sämtlichen Verkehr im lokalen Netz zu lauschen und somit eine Layer2-Verbindung zum virtuellen Netz verfügbar zu machen.

Management Interfaces definieren Punkt-zu-Punkt Verbindungen zwischen dem Host und den virtuellen Maschinen und besitzen deshalb immer die Netzmaske /30. Die Schnittstelle im Host hat den Namen *name-eth0*, wobei *name* dem Namen der virtuellen Maschine entspricht (deklariert über das Attribut “name“ des <vm>-Tag). Innerhalb der virtuellen

Maschine heißt die entsprechende Schnittstelle immer *eth0*, alle anderen darin definierten Schnittstellen werden also ab *eth1* aufsteigend benannt. Jeder Management-Schnittstelle wird eine IPv4-Adresse zugewiesen. Dies beginnt standardmäßig bei 192.168.0.1. Wird durch den Tag <ip_offset> ein Offset angegeben, wird dieser vorher addiert (siehe Kapitel XML Syntax und Semantik).

UML Interfaces werden durch den Tun/Tap-Gerätetreiber als Gegenstück für jede Schnittstelle einer virtuellen Maschine auf dem Host angelegt. Im Falle von virtuellen Netzen, die als switch-Daemon laufen, sind solche Schnittstellen nicht nötig, da die Netzwerkkommunikation auf Ebene der Interprozesskommunikation stattfindet (siehe Kapitel Switch Daemon). Bezeichnet werden diese Schnittstellen mit *name-ethN*, wobei *name* dem Namen der virtuellen Maschine entspricht und *N* einer Integer Zahl (ausser 0 - siehe Management Interfaces), welche im "id"-Attribut innerhalb des <if>-Tag deklariert ist. Diese Schnittstellen haben keine IP-Adresse, sie werden verwendet um sie an die virtuelle Bridge zu binden und so ein Ethernet zwischen den VM aufzubauen. Das Gegenstück zu dieser Schnittstelle in der virtuellen Maschine heißt *ethN*.

Betrachtet man diese beiden Schnittstellen als „Enden eines Netzwerkkabels", so verläuft der Aufbau einer Ethernet-Topologie im übertragenen Sinne folgendermaßen: *ethN* in der VM ist das eine Ende des Kabels, erreichbar über die in der VM konfigurierte IP-Adresse. Das andere Ende des Kabels ist die dazugehörige Tap-Schnittstelle *name-ethN* im Host, ohne IP-Adresse aber mit Mac-Adresse. Dieses Ende wird in die virtuelle Bridge „eingestöpselt". Da man nahezu beliebig viele Instanzen solcher UML-Interfaces „einstöpseln" kann, können in gewohnter Weise beliebige Ethernet-Netze zusammengesteckt werden.

Virtual Network Interfaces treten immer in Zusammenhang mit als „virtual_bridge" deklarierten virtuellen Netzen auf. Sie haben die Bezeichnung *name*, welche innerhalb des <net>-Tag mit dem name-Attribut definiert wird. Diese Schnittstellen sind die Implementation der „virtual_bridge". Jede Instanz einer „virtual_bridge" wird durch eine Schnittstelle im Host repräsentiert. Dadurch kann die Bridge insbesondere mit den Parametern `up` und `down` hoch- bzw. heruntergefahren werden und nach Zuteilung einer IP-Adresse und Subnetzmaske auch als Verbindungspunkt zum externen Netzwerk dienen. (siehe Kapitel Virtual Bridging)

2.4 Virtuelle Netze

Virtuelle Netze können durch den VNUML-Parser durch zwei verschiedene Transporttypen für Ethernet-Netze aufgebaut werden. Dazu werden das frei verfügbare Bridge-Modul oder der switch-Daemon verwendet. Das Bridge-Modul läuft im Kernelspace und benötigt root-Rechte, da über den Tun/Tap-Gerätetreiber die Schnittstellen-Konfiguration des Host verändert wird. Der switch-Daemon läuft im Userspace und benötigt keine root-Rechte, da virtuelle Maschinen hier direkt auf Prozessebene kommunizieren, ohne die Host-Konfiguration ändern zu müssen. Da VNUML jedoch, um den Vorteil der user-Rechte beim switch-Daemon nicht zu verlieren, auf eine Hostanbindung des switch-Daemon verzichtet und somit auch keine externen Verbindungen möglich sind, soll hierauf nicht ausführlich eingegangen werden.

2.4.1 Virtual Bridging

Eine Bridge leitet Verkehr auf Schicht 2 des OSI-Modells zwischen Netzsegmenten weiter. Sie wird hauptsächlich eingesetzt, um ein Netz in verschiedene Kollisionsdomänen aufzuteilen. Somit kann die Last in großen Netzen vermindert werden, da jeder Netzstrang nur die Pakete empfängt, deren Empfänger sich auch in diesem Netz befindet.[3]

Die Netze die durch eine Bridge verbunden sind, bilden ein größeres logisches Netz. Die Arbeit die durch die Bridge verrichtet wird, soll auf IP-Ebene völlig transparent bleiben, in einem `traceroute` kommt somit keine Bridge vor.

Im Linux-Kernel ist **bridging** seit Version 2.4 vollständig implementiert. Dies ermöglicht das Starten einer *virtual_bridge* innerhalb eines Linux-Hostsystems. Da diese als Schnittstelle repräsentiert wird, sind für die Ausführung der Kommandos root-Rechte nötig. Um die Algorithmen administrieren zu können, wird das Paket „bridge-utils“ benötigt. Darin enthalten ist das Kommando `brctl`:

brctl addbr *bridgename* Erzeugt eine logische Instanz einer Bridge mit dem Namen *bridgename*. Die Bridge kann als Container für die ihr hinzugefügten Schnittstellen interpretiert werden, welche in einem größeren logischen Netz liegen. Jede Instanz wird durch eine neue Schnittstelle im Host repräsentiert.

brctl delbr *bridgename* Beenden und Löschen einer Instanz.

brctl addif *bridgename device* Fügt die Schnittstelle mit dem Namen *device* zur Bridge *bridgename* hinzu. Nach kurzer Zeit lernt die Bridge die MAC-Adresse und kann Pakete gezielt weiterleiten.

brctl delif *bridgename device* Entfernt die Schnittstelle mit dem Namen *device*.

brctl show Zeigt eine Zusammenfassung aller aktiven Instanzen und der aktuell eingebundenen Schnittstellen.

brctl showmacs *bridgename* Zeigt alle der Bridge bekannten MAC-Adressen und deren Timer.

brctl stp *bridgename* on/off Aktiviert/Deaktiviert das Spanning Tree Protocol (STP), welches benötigt wird um bei redundant vernetzten Bridges Endlosschleifen zu vermeiden.

In der VNUML-Version 1.5 sind standardmäßig alle in der Simulation erzeugten Schnittstellen über solche *virtual_bridges* verbunden. Dazu wird für jede Schnittstelle einer VM ein tap-device ohne IP-Adresse (siehe UML Interfaces) auf dem Host angelegt, welcher dann an die für das jeweilige Netz erzeugte *virtual_bridge* gebunden wird. Durch diesen „Trick“ muss der Verkehr zwischen den virtuellen Maschinen nicht auf IP-Schicht durch den Host geroutet werden: Ist auf dem Host IP-Forwarding aktiv, würde die reine Anbindung der VM via tap-devices genügen um eine Kommunikation zu anderen VMs herzustellen, da Vehrker durch den Host an die jeweilige Schnittstelle weitergeleitet würde, sofern die erzeugten tap*N* Schnittstellen auf dem Host eine IP-Adresse besitzen. Dies würde jedoch auf Layer3 einen zusätzlichen Hop bedeuten (VM1 - Tap1 - VM2). Aus diesem Grund verlagert VNUML die Vernetzung auf Layer2 und verwendet dafür eine direkte Anbindung der tap-devices an die oben vorgestellte *virtual_bridge* (diese Anbindung kann als ein „einstecken“ eines Ethernet-Kabels an ein Layer2-Gerät interpretiert werden). Da tap-devices Ethernet-Frames verwenden, ist über die Bridge nun eine vollständige Verbindung auf Layer2 hergestellt.
Ein `traceroute` zeigt, dass zwei virtuelle Maschinen die via tap-devices an eine *virtual_bridge* gebunden sind, auf Layer3 direkt (ohne zusätzlichen Hop) verbunden sind. Da beliebig viele Instanzen solcher *virtual_bridges* erzeugt und bis zu 255 (Kernel 2.4) bzw. 1023 (Kernel 2.6) Schnittstellen an jede einzelne gebunden werden können, ist die Simulation (fast) beliebiger Ethernet-Topologien möglich.
Der Aufbau der virtuellen Netze ist für den Benutzer transparent und wird durch den VNUML-Parser anhand der XML-Spezifikation übernommen. Externe Verbindungen sind mit VNUML nur über diesen Transporttyp möglich, und die entsprechenden Netze müssen mit diesem Typ deklariert werden:
`<net name='extern' mode='virtual_bridge' external='eth0'/>`.
Die physikalisch vorhandene Schnittstelle *eth0* wird also dem Bridge-Container „extern“ hinzugefügt, wodurch das Netzsegment ausserhalb des Hostsystem an *eth0* mit Hilfe der Bridge ein logisches Netz mit den virtuellen Netzsegmenten innerhalb des Hostsystems bildet.

2.4.2 Switch Daemon

Als zweite Variante unterstützt VNUML den Transporttyp switch-Daemon, mit dem ein komplett virtuelles Netz mit Hilfe des Moduls *uml_switch* erstellt wird:
<net name='Net1' mode='uml_switch'/>.
Eine Anbindung an das Host-Netzwerk ist zwar via tap-devices auch beim *uml_switch* möglich, wird von VNUML aber nicht unterstützt, um den Vorteil der user-Rechte nicht zu verlieren. In Version 1.6 kommt diesem Transporttyp eine größere Bedeutung zu, da in dieser Programmversion versucht wird, Simulationen auch ohne root-Rechte durchführbar zu machen.

Das Modul *uml_switch* ist ein in Software implementierter Ethernet-Switch. Es läuft im Userspace, erreicht durch die nötigen Kontextwechsel in den Kernelspace jedoch nicht die Performanz der *virtual_bridge*. Der Daemon wird als Benutzer mit user-Rechten über die Kommandozeile gestartet (z.B. `$ uml_switch -unix /tmp/switch1.sock`). Dabei muss ein UNIX-Socket spezifiziert werden. Dieser Socket wird beim Start dem entsprechenden UML-Prozess übergeben. Der Aufbau eines virtuellen Switch-Netzwerks wird durch den VNUML-Parser übernommen und bleibt für den Benutzer transparent. Stürzt ein switch-Daemon ab, müssen alle angebundenen VM neugestartet werden, um die Socket-Verbindung neu zu initialisieren.

3 XML Syntax und Semantik

In diesem Kapitel werden alle für interne und externe Verbindungen relevanten XML-Tags der VNUML Sprache behandelt. Für eine ausführliche Beschreibung aller vorhandenen Tags siehe [1]. Die vier Haupt-Tags 1.Stufe sind

<**global**> beschreibt allen globalen Elemente der Simulation

<**net**> beschreibt virtuelle Netzwerke

<**vm**> spezifiziert die virtuellen Maschinen

<**host**> Konfiguration des Host (optional).

Die darin für diese Arbeit relevanten Tags werden im Folgenden genauer spezifiziert.

3.1 Innerhalb des <global>-Tag:

<**automac**> Dieses Tag bewirkt die automatische Erzeugung von MAC- Adressen. Die Verwendung ist optional. Wird <automac> verwendet, müssen die MAC-Adressen der virtuellen Rechner nicht mehr von Hand über das <mac>-Tag gesetzt werden. Die Angabe eines offsets ist über das Attribut "offset" möglich. Dies ist sinnvoll, wenn mehrere Simulationen mit automatischen MAC-Adressen gleichzeitig laufen und interagieren sollen. Die Verwendung dieses Tags wird empfohlen um doppelten MAC-Adressen vorzubeugen. Wird innerhalb des <if>-Tags durch <mac> eine spezifische Adresse zugewiesen, obwohl global <automac> aktiv ist, wird hierbei vom Parser die selbst definierte Adresse gewählt. In diesem Fall sollte besonders auf die Eindeutigkeit geachtet werden, damit durch <automac> nicht eine schon existierende Adresse generiert wird. Adressen werden in folgendem Format generiert: *fe:fd:0:Z:X:Y*, wobei X die Nummer der virtuellen Maschine ist (aufsteigend nach Definition in der XML-Datei, beginnend bei 1) und Y die Nummer der Schnittstelle (definiert über das id-Atrribut im <if>-Tag). Z ist der Wert des offset (0 falls nicht angegeben).
Für verteilte Simulationen muss immer ein offset angegeben werden, um doppelte MAC-Adressen zu vermeiden. Dabei sollte in den XML-Dateien auf jedem Host aufsteigend ein genügend großer offset angegeben werden (HostA: <`automac offset='10'/`>, HostB: <`automac offset='20'/`>).

<**ip_offset**> Über dieses optionale Tag kann ein offset definiert werden, der auf die Management-Interfaces (siehe Kapitel 2.3) der virtuellen Rechner addiert wird. Der default-offset

ist 0. Anhand des Attributs prefix kann ein Prefix für IPv4-Adressen angegeben werden, die auf dem Host für die virtuellen Rechner generiert werden müssen. Es ist optional und wird standardmäßig auf 192.168. gesetzt. Das Prefix umfasst stets die ersten zwei Byte der IP-Adresse. Somit wird mit prefix immer ein Klasse B Netz angegeben. Laufen mehrere Szenarien gleichzeitig auf einem Host muss ein ip_offset angegeben werden, um Überschneidungen zu vermeiden.
In verteilten Simulationen ist kein offset erforderlich, da die Management-Verbindungen nur Punkt-zu-Punkt Verbindungen sind (/30 Netz) und sich gegenseitig -über mehrere Rechner hinweg- nicht beeinflussen.

<**netconfig**> Dieses optionale Tag ermöglicht das An- und Ausschalten des Spanning Tree Protokolls (STP) für die virtuellen Bridges, die von VNUML erzeugt werden. Desweiteren kann der promiscuous Mode für die teilnehmenden Schnittstellen aktiviert oder deaktiviert werden. Standardmäßig gilt: stp="off" und promisc="on".

<**tun_device**> Über dieses optionale Tag kann ein alternatives Device zum Erzeugen von virtuellen Netzwerk- Schnittstellen angegeben werden. Das default-Device ist */dev/net/tun*.

3.2 Innerhalb des <net>-Tag:

Dieses Tag bietet keine Tags zum Einstellen weiterer Optionen (ausser <bw> falls „ppp" genutzt wird, siehe Online-Referenz) und dient lediglich dem Einführen eines virtuellen Netzes mit einem Namen. Die IP-Adressen, die in einem Netz vorhanden sind, ergeben sich aus den Schnittstellen der virtuellen Rechner, die mit einem Netz verbunden sind. Um ein Netzwerk identifizieren zu können, muss ein eindeutiger Name über das Attribut „name" vergeben werden. Version 1.5 bietet zwei mögliche Netzwerke an, die über das Attribut „mode" gesetzt werden können:

virtual_bridge Durch diesen Wert wird das deklarierte Netz mit einer virtuellen Bridge implementiert. Zum Erzeugen eines solchen Netzwerks werden root-Rechte benötigt. Nur für diese Art Netz kann das Attribut „external" angegeben werden, welches das Verbinden der virtuellen Bridge mit einer realen Schnittstelle des Hosts ermöglicht. Auf diese Weise kann ein virtuelles Netzwerk mit dem Internet verbunden werden. Falls die angegebene Host-Schnittstelle einem VLAN angehört, kann über das Attribut „vlan" die Nummer des VLAN angegeben werden, in dem sich die Schnittstelle befindet.

uml_switch Durch die Angabe dieses Werts wird ein UML-Switch zum Verbinden der virtuellen Rechner des Netzes erzeugt. Für solche Netzwerke sind keine root-Rechte notwendig.

Möchte man ein Hub anstatt einem Switch verwenden, kann über das Attribut „hub“ der Switch in ein Hub umgewandelt werden (hub = “yes“). Der Defaultwert für dieses Attribut ist “no“.

Der Defaultwert von „mode“ ist “virtual_bridge“. Es können nahezu beliebig viele Netze erzeugt werden. Die einzige Begrenzung besteht in der Tatsache, dass die Länge der Netznamen auf sieben Buchstaben begrenzt ist.
In verteilten Simulationen muss der Name des Netzes mit externer Anbindung nicht auf allen Hostsystemen identisch sein. Um die Lesbarkeit zu erhöhen, empfiehlt es sich jedoch solche Netze entsprechend zu bezeichnen (z.B. <net name='extern' external='eth0'/>).
Eine (physikalische) Schnittstelle kann nur an eine Bridge gebunden werden. Das impliziert, dass nicht für jedes Netz der Topologie, welches über eine externe Verbindung laufen soll, ein eigenes externes Netz angelegt werden muss. Ein Netz (mit dem Attribut „external“) beinhaltet logisch alle Netze, die auf der externen Verbindung zwischen zwei Hostrechnern liegen. (siehe Kapitel 6.1.2).

3.3 Innerhalb des <vm>-Tag:

<**if**> Mit diesem Tag werden Netzwerk-Schnittstellen auf dem virtuellen Rechner angelegt. Das Attribut „id='*N*'“ gibt die Integerzahl *N* an, die später den Namen der entsprechenden Schnittstelle (eth*N*) definiert. Diese Zahl muss größer als 0 sein (0 ist für das Management-Interface reserviert). Im virtuellen Rechner *name* wird dann die Schnittstelle eth*N* angelegt, im Host wird eine entsprechende Tun/Tap-Schnittstelle mit dem Namen *name*-eth*N* angelegt (siehe UML-Interfaces). Mit dem Attribut „net“ wird das Netzwerk angegeben, mit dem die neue Schnittstelle verbunden werden soll. Der Name muss einem zuvor angelegten Netz (im Attribut „name“ des <net>-Tag) entsprechen. Bei externen Verbindungen wird an dieser Stelle identisch verfahren, es genügt die Angabe des Namens des vorher als „external“ deklarierten Netzes. Damit wird diese Schnittstelle einer VM automatisch an die Bridge gebunden, die auch die physikalische Host-Schnittstelle enthält.

Innerhalb von <if> können drei weitere Tags benutzt werden: <mac>, <ipv4>, <ipv6>. Diese können auch bei verteilten Simulationen in gewohnter Weise eingesetzt werden, Syntax und Semantik bleiben identisch.

3.4 Innerhalb des <host>-Tag:

Die Host-Konfiguration sollte nur durchgeführt werden, wenn der Host ein Teil der Simulation ist oder bei externen Verbindungen die eigene Konnektivität zum Internet oder lokalen Netz erhalten bleiben soll. Das <host>-Tag funktioniert analog zum <vm>-Tag (alle für eine VM vorhandenen Tags können auch hier verwendet werden). Mit den Tags <route> und <forwarding> kann eine Konfiguration des Host angegeben werden, die **während** der Simulation gelten soll (z.B. default-Route definieren und IP-Forwarding aktivieren). Diese müssen explizit angegeben werden, da die Linux-Konfiguration der Netzwerk-Devices verloren geht, sobald der Host bzw. eine Schnittstelle des Host an der Simulation teilnehmen. Die beiden folgenden Tags sind zusätzlich innerhalb des <host>-Tag vorhanden:

<**hostif**> Dieses Tag ist das Pendant zum <if>-Tag. Es definiert eine Schnittstelle und das daran liegende Netz. Die neue Schnittstelle wird auf dem Host automatisch erzeugt und hat stets den Namen des virtuellen Netzes, mit dem sie verbunden ist (ist das Attribut „net“ auf Net0 gesetzt (`net='Net0'`), so heißt diese Schnittstelle im Host: Net0). Die Schnittstelle ist also keine neu erzeugte Tun/Tap-Instanz, sondern die Schnittstelle der *virtual_bridge* selbst, die das Netz implementiert an welches diese Schnittstelle angebunden werden soll. Durch diesen Trick ist es möglich gleichzeitig über die physikalische Schnittstelle ein virtuelles Netz auf Layer2 mit Hilfe einer Bridge an das externe Netz anzubinden und, da die Bridge eine IP-Konfiguration erhält, die Konnektivität des Host auf Layer3 an das externe Netz zu erhalten. Dieses Tag hat kein id-Attribut. Das Tag <mac> ist innerhalb des <hostif>-Tag nicht verfügbar. Die Bedeutung der Tags <ipv4> und <ipv6> ändert sich nicht.

Da jegliche Konfiguration der physikalischen Schnittstelle durch die Verbindung mit der virtuellen Bridge verloren geht, können virtuelle Netze, deren external-Attribut auf eine Schnittstelle des Hosts gesetzt wurde, mit diesem Tag konfiguriert werden. Ist die Konfiguration identisch zu der der pyhsikalischen Schnittstelle, bleibt der Host zum externen Netz verbunden. Seine Schnittstelle zu diesem ist nun die Bridge mit der entsprechenden IPv4-Adresse, die echte Schnittstelle leitet den Verkehr auf Layer2 zu dieser Bridge. Um diese Schnittstelle wie vor dem Start der Simulation weiter nutzen zu können, müssen die vorherigen Einstellungen mit den <ipv4>- bzw. <ipv6>-Tags wieder eingetragen werden. Eventuelle Routingeinträge und Standardgateways müssen innerhalb des <host>-Tags mit <route> wiederhergestellt werden.

<**physicalif**> In diesem Tag kann die ursprüngliche Konfiguration einer physikalischen Schnittstelle abgespeichert werden, da deren Konfiguration verloren geht sobald man sie an eine *virtual_bridge* bindet. Anhand dieses Tag kann der VNUML-Parser während dem

Herunterfahren des Szenarios (-d Mode) die Schnittstellenkonfiguration wiederherstellen. Die Konfiguration wird anhand der Attribute „type“ (ipv4 oder ipv6), „name“ (z.B. eth0), „ip“ (IPv6-Adressen mit Angabe der Maske), „mask“ (nur für IPv4-Adressen) und „gw“ (Standardgateway) gesetzt. Standardgateway ist zwar keine Schnittstellenkonfiguration, sondern Routing-Information, diese Information geht jedoch verloren sobald die entsprechende Schnittstelle ihre Konfiguration verliert. Anhand des Attributes „gw“ kann der Parser diese Routing-Information wiederherstellen. Für jede Schnittstelle können maximal zwei dieser Tags definiert werden (jeweils eins für IPv4 und IPv6). Beispiel: im Netz der Uni-Koblenz sichert diese Deklaration die IPv4-Konfiguration der Schnittstelle *eth0*: `<physicalif name='eth0' ip='141.26.71.18' mask='255.255.248.0' gw='141.26.64.9'/>`.

4 Hardwareanforderungen

Mit großen Topologien steigt auch der Anspruch an die Hardware des Hostrechners. Da keine Bandbreiten für die Netze angegeben werden können, ist es nicht sinnvoll VNUML für Untersuchungen bezüglich Datenverkehrsraten o.ä. heranzuziehen. Vielmehr ist der Hauptrechenaufwand erledigt, sobald die Topologie und danach die darauf laufende Simulation gestartet sind. Die dann folgenden Untersuchungen bleiben meist im Rahmen von Protokollverhalten, Erreichbarkeiten, Routenaustausch etc. und stellen keine besonderen Anforderungen an die Hardware. Da ein Echtzeitverhalten vieler virtueller Maschinen auf dem Host ohnehin nicht garantiert werden kann, sind kleine Unterschiede in Signallaufzeiten (z.B. Zeitangaben bei ping in ms) nicht von Bedeutung. Der umfangreichste Rechenaufwand ist also der Startvorgang des Szenarios selbst, also insbesondere das Hochfahren jeder einzelnen virtuellen Maschine.

Dieser Aufwand muss in verschiedenen Anwendungsbereichen unterschieden werden: Wird ein Szenario in einer extra dafür vorgesehenen Laborumgebung gestartet, in der die Topologie **dauerhaft** erhalten bleiben kann und wird nur die darauf aufsetzende Simulation während dem Betrieb verändert (Simulationen können z.B. nach Konfigurationsänderungen durch den Parser neugestartet werden), so sind Wartezeiten für den initialen Startvorgang des Szenarios von bis zu 60 Minuten (=3600 Sekunden) durchaus akzeptabel. Werden Szenarios jedoch am eigenen, bzw. an einem Rechner an dem viele Personen arbeiten, gestartet, so muss das komplette Szenario aus Performance- bzw. Konfigurationsgründen immer komplett heruntergefahren und neu gestartet werden. Hierbei sind Wartezeiten über 10 Minuten (=600 Sekunden) in den meisten Fällen nicht gewünscht. Die folgende Tabelle soll eine grobe Orientierung geben, bei welcher Hardwareausstattung und ab welcher Anzahl virtueller Maschinen das Aufsplitten eines Szenarios auf mehrere Rechner sinvoll wird. Die verwendeten Rechner in den Testläufen sind:

Notebook Centrino 1.5GHz 1024MB-RAM + 512MB-Swap

Becks Pentium4 2.4GHz 512MB-RAM + 1024MB-Swap

Licher Celeron 500MHz 128MB-RAM + 256MB-Swap

Nitrogen Pentium2 400MHz 256MB-RAM + 512MB-Swap

Anzahl VM	Notebook	Becks	Licher	Nitrogen
1	15 sek	19 sek	40 sek	86 sek
2	25 sek	33 sek	76 sek	131 sek
3	36 sek	46 sek	99 sek	187 sek
4	44 sek	64 sek	141 sek	297 sek
5	53 sek	79 sek	170 sek	343 sek
6	64 sek	92 sek	218 sek	366 sek
7	77 sek	112 sek	279 sek	588 sek
8	85 sek	125 sek	362 sek	657 sek
9	97 sek	139 sek	523 sek	870 sek
10	108 sek	157 sek	692 sek	1178 sek
15	155 sek	238 sek	5581 sek*	1178 sek
20	211 sek	336 sek	x	2894 sek
25	264 sek	484 sek	x	4996 sek*
30	322 sek	558 sek	x	x
40	435 sek	1121 sek	x	x
50	542 sek	x	x	x

Tabelle 1: Bootdauer der Szenarios in Sekunden

Das Suffix * bedeutet, dass das Szenario zwar ordnungsgemäß startet, das System aber so ausgelastet ist, dass weiteres Arbeiten nicht mehr möglich ist. Zusätzlich ist die Zeit aufzuaddieren, die das anschließende Starten der Simulation (über den Parameter -x start@...) benötigt. Sind ein Großteil der virtuellen Maschinen Router auf denen Zebra/Quagga Daemons gestartet werden, so entspricht diese Zeit nocheinmal **mindestens der halben Bootdauer**.
Deutlich zu erkennen ist, dass der verfügbare Hauptspeicher (RAM+Swap) ausschlaggebend ist und ab einer bestimmten Szenariogröße auch die Grenze bildet: Licher ist nach dem Start von 15 VM nicht mehr ansprechbar, Nitrogen erst bei 25 VM. Ist ausreichend Speicher vorhanden (Notebook), so lässt sich folgende Regel aufstellen: „**Anzahl virtueller Maschinen x 11 Sekunden = Bootdauer**“.
Alles in Allem kann diese Tabelle nur eine grobe Richtung aufzeigen, da die Bootdauer sehr abhängig vom jeweils erstellten Netz (Anzahl Netze, Anzahl Interfaces) und vom verwendeten Dateisystem bzw. dessen Mount-Strategie (z.B. Copy-On-Write) ist.

Zum konkrekten Vergleich eignet sich die im Kapitel 6.3 beschriebene Topologie *bignet*. Diese besteht aus 51 virtuellen Maschinen in insgesamt 6 Autonomen Systemen (AS1-AS6), von denen insgesamt 23 als Router fungieren, d.h. für jeden Routertyp muss ein entsprechender Daemon gestartet und konfiguriert werden. Auf dem Rechner *Notebook* (s.o.) benötigt der Aufbau des Szenarios 639 Sekunden, der anschließende Start der Daemons (über den Parameter -x start@...) 786 Sekunden. Das Herunterfahren benötigt für den Stop der Simulation 660 Sekunden und für den Abbau der Topologie 560 Sekunden. Verteilt man das Beispielnetz auf vier Hostrechner (siehe Anhang E), ergeben sich folgende Zeiten:

AS auf Hostrechner	Szenario	Simulation
AS4	285 sek	376 sek
AS5	154 sek	168 sek
AS6	151 sek	168 sek
AS1-3	250 sek	50 sek

Tabelle 2: Bootdauer des verteilten Netzes

Lässt man die (einmalig benötigte) Zeit ausser Acht, die die Umwandlung eines normalen in ein verteiltes Netz in Anspruch nimmt, so sind hier die längsten Zeiten ausschlaggebend, da man auf jedem Rechner zur gleichen Zeit mit dem Hochfahren beginnen kann. Die beiden längsten Zeiten addieren sich zu 661 Sekunden (ca. 11 Min.), die Gesamtzeit beim Start auf einem Rechner (s.o.) beträgt 1425 Sekunden (ca. 24 Min.). Das entspricht einer Zeitersparnis von ca. 55 Prozent.

Anmerkung: Deutliche Auswirkung auf die Bootdauer hat der Filesystemcheck, der z.B. bei dem von VNUML mitgelieferten Dateisystem alle 30 Tage bzw. 180 Mounts ausgeführt wird. Bei alten Dateisystemen wird dieser Check automatisch jedesmal und für jede VM durchgeführt, was die Bootdauer nocheinmal verdoppelt. Abhilfe schafft hier das Kommando `tune2fs` mit dem sich die Parameter für Dateisystem-Images setzen lassen (`tune2fs -i 0 -c 0` verhindert den Filesystemcheck dauerhaft).

5 Szenarios mit externer Verbindung

Um die in den <net>-Tags definierten Netze zu implementieren erstellt VNUML virtuelle Bridges, an die die Schnittstellen der virtuellen Maschinen gebunden werden. Diese Netze ermöglichen einen Datenverkehr innerhalb der laufenden Simulationen. Wie in Kapitel 2.4.1 erläutert, werden solche virtuellen Netze standardmäßig mit Hilfe der „bridge-utils“ angelegt. Dabei ist es auch möglich an die virtuelle Bridge, neben virtuellen Schnittstellen, echte Schnittstellen des Hostrechners zu binden. Hierzu wird lediglich vom Parser, über den entsprechenden Bridge-Befehl, die reale Host-Schnittstelle *ethN* zum Netz, welches mit „external“ deklariert wurde, hinzugefügt. Die Schnittstelle und damit auch das Netz an dieser sind somit schon ein Teil der Simulation. Die Umsetzung dieser Schnittstellen-Verbindung innerhalb des Host findet auf Layer2 statt und ist für den Benutzer völlig transparent.

Im Folgenden soll der Entwurf und die Implementation eines Szenarios mit einer externen Verbindung über die Netzwerkkarte des Host gezeigt werden. Das hier verwendete Beispielszenario (*externnormal.xml*) besteht aus 6 virtuellen Maschinen die in 3 Netzen liegen und verdeutlicht das Anbinden der Simulation an ein LAN mit Internetgateway:

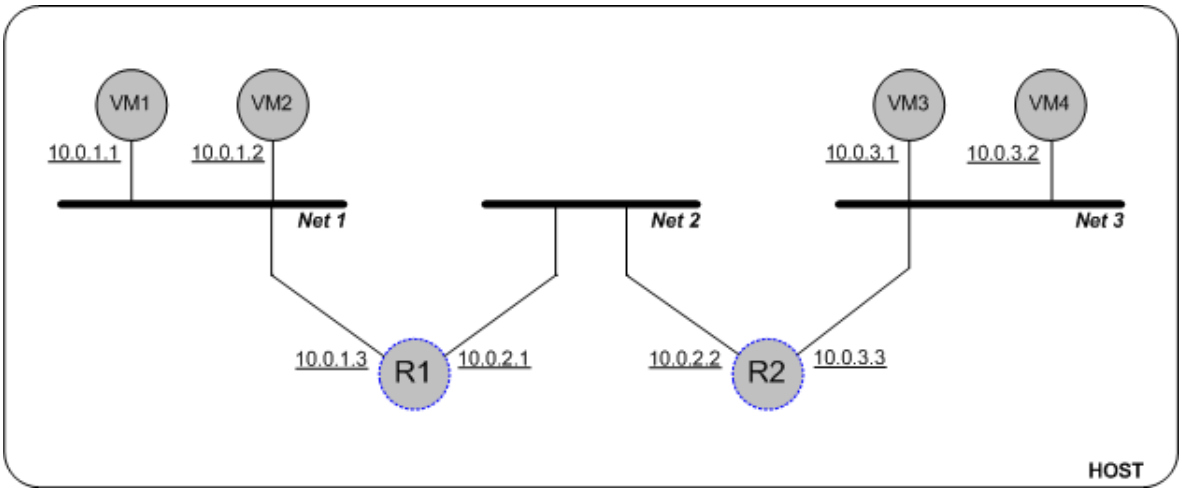

Abbildung 1: Beispielszenario externnormal.xml (XML Beschreibung auf der CD)

R1 liegt in Net1 und Net2, R2 liegt in Net2 und Net3. Die Erreichbarkeiten, insbesondere für R1 nach Net3 und für R2 nach Net1 sind, um unnötige Komplexität des Beispiels zu vermeiden, nicht durch Routingprotokolle verfügbar, sondern durch statische Routen implementiert. Zusätzlich ist auf R1 und R2 *ip-forwarding* aktiviert.
Um externe Netze für die Rechner in der Simulation erreichbar zu machen, lassen sich zwei Wege unterscheiden:

5.1 Anbindung auf Layer2

Über R1 soll eine direkte Verbindung über die Schnittstelle *eth0* des Hostsystems zu einem Netz ausserhalb der Simulation hergestellt werden. Hierzu muss ein neues Netz deklariert werden, um über die so erzeugte virtuelle Bridge eine zusätzliche Schnittstelle von R1 mit der realen Schnittstelle *eth0* zu verbinden. R1 muss also eine weitere virtuelle Schnittstelle (*eth3)* innerhalb der XML-Datei hinzugefügt werden:

```
<if id="3" net="extern">
 <mac>00:0E:35:24:F8:2A</mac>
 <ipv4 mask="255.255.255.0">192.168.2.101</ipv4>
</if>
```

Die IP-Adresse dieser Schnittstelle muss einer im externen Netz verfügbaren entsprechen, da später der Verkehr direkt über R1 in die laufende Simulation kommen soll. Läuft im LAN ein DHCP Server, muss diesem die Kombination aus automatisch erzeugter MAC-Adresse und dazugehöriger IP-Adresse bekannt gemacht werden, oder es muss -wie oben beschrieben- eine, dem DHCP-Server bekannte Kombination aus MAC-Adresse und IP-Adresse, vorgegeben werden. Das durch *net=“extern“* angebundene Netz muss noch im Kopf der XML-Datei definiert werden:

```
<net name="extern" external="eth0"/>
```

Durch das Attribut “external“ wird der Parser angewiesen an das Netz, neben den virtuellen Schnittstellen, auch die Schnittstelle *eth0* des Hostsystems einzubinden. Damit von innerhalb der Simulation alle Rechner im LAN erreichbar werden, muss auf R1 eine default-Route zum Standardgateway angelegt werden:

```
<route type="inet" gw="192.168.2.1">default</route>
```

Diese default-Route ermöglicht es den Rechnern innerhalb der Simulation auch Ziele ausserhalb des LAN zu erreichen. Die folgende Darstellung zeigt die aktuelle Topologie. Um den Sachverhalt zu verdeutlichen sind in diesem Bild die Netze anhand ihrer virtuellen Bridges dargestellt:

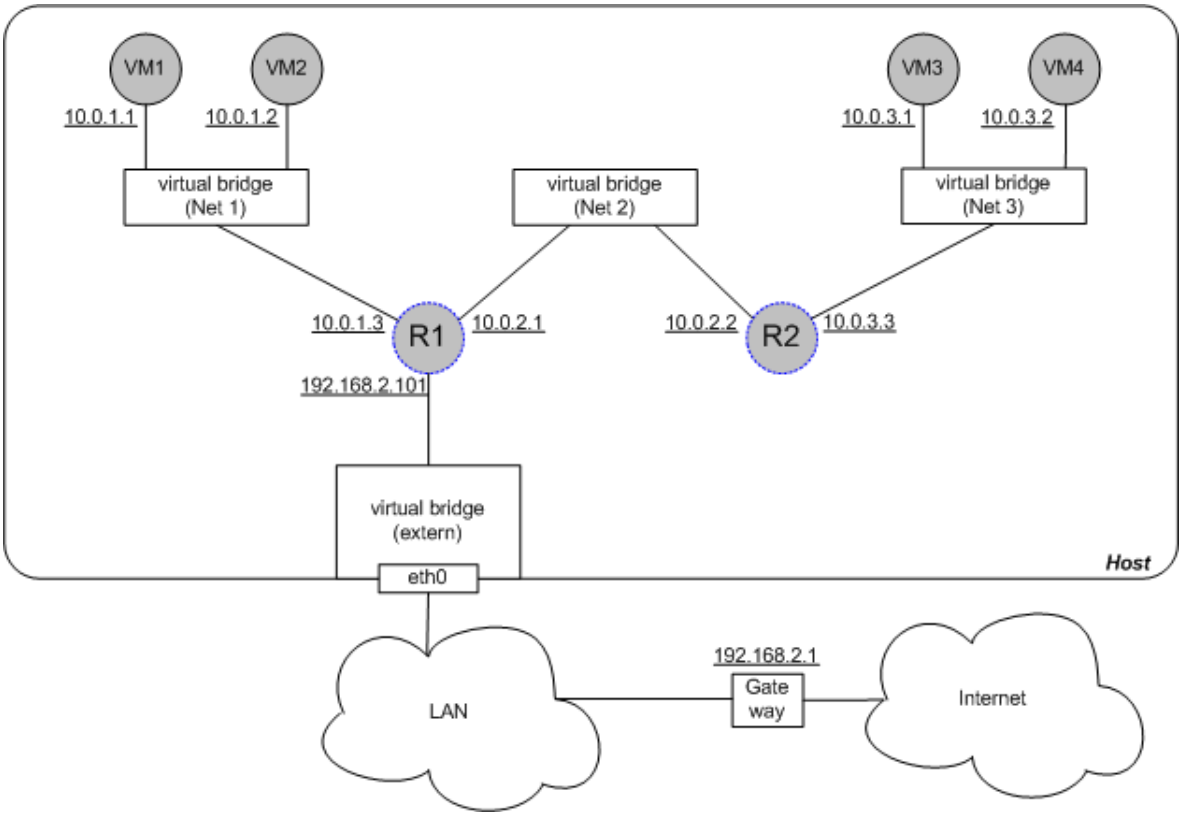

Abbildung 2: Beispielszenario externlayer2.xml (XML Beschreibung siehe Anhang A)

Ein Aufruf des Befehls `brctl show` in der Kommandozeile verdeutlicht die Verbindungen in den Netzen innerhalb und ausserhalb der Simulation:

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
extern          8000.000e3524f82a       no              eth0
                                                        R1-eth3
Net1            8000.00ff3974c489       no              VM1-eth1
                                                        VM2-eth1
                                                        R1-eth1
Net2            8000.00ff25701933       no              R1-eth2
                                                        R2-eth1
Net3            8000.00ff5e9ed13c       no              VM3-eth1
                                                        VM4-eth1
                                                        R2-eth2
```

Die Bridge *extern* ist mit *eth0* (Host) und *eth3* (R1) verbunden. Die *id* jeder Bridge ergibt sich aus der ersten ihr angeschlossenen MAC-Adresse. Die Ausgabe der Routentabelle auf der

virtuellen Maschine R1 bestätigt, dass diese nun eine direkte Verbindung (über Tap-Schnittstelle *eth3*) auf Layer2 zum Gateway (192.168.2.1) des externen Netzes hat:

```
R1:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask          Flags  Iface
192.168.0.16    0.0.0.0         255.255.255.252  U      eth0
192.168.2.0     0.0.0.0         255.255.255.0    U      eth3
10.0.1.0        0.0.0.0         255.255.255.0    U      eth1
10.0.2.0        0.0.0.0         255.255.255.0    U      eth2
10.0.3.0        10.0.2.2        255.255.255.0    UG     eth2
0.0.0.0         192.168.2.1     0.0.0.0          UG     eth3
```

Trotzdem bleibt nun ein Ping von VM1 an 192.168.2.100 (zweiter Rechner im LAN - angeschlossen am selben Switch wie der Host) unbeantwortet. Auch ein Ping von diesem Rechner zurück in die Simulation zu VM1 bleibt erfolglos. Der Grund hierfür liegt in der Art der Anbindung an das externe Netz. Die virtuellen Maschinen der Simulation mit Adressen aus dem privaten Bereich 10.0.0.0/16, existieren nur lokal „hinter“ R1 und sind nicht einzeln (z.B. über automatische Adressvergabe per DHCP) an das LAN angeschlossen. Die Lösung dieses Problems liegt in der Verwendung einer Form von NAT (Network Adress Translation), welches als IP-Masquerading im Linux-Kernel ab Version 2.2.x durch das Netfilter-Framework implementiert ist.

IP-Masquerading ist eine Netzwerkfunktion unter Linux, ähnlich der eine-zu-vielen Übersetzungen von Netzwerkadressen, die häufig in kommerziellen Firewalls und Netzwerkroutern anzutreffen sind. Wenn, zum Beispiel, ein Linux-Host mittels PPP, Ethernet, etc. mit dem Internet verbunden ist, erlaubt die Masquerading-Funktion anderen, intern mit dem Linux-Rechner verbundenen, Computern die Nutzung des Internet. Masquerading bietet diese Funktionalität sogar, wenn die internen Maschinen keine offiziell zugeteilten Internet-IP-Adressen haben. Es erlaubt einem ganzen Satz von Maschinen unsichtbares betreten des Internet, versteckt hinter einem Gateway-System, welches nach außen hin als ein einzelnes System erscheint.[4] Dieses Masquerading wird auf der virtuellen Maschine R1 für über *eth3* ausgehende Verbindungen gesetzt: `iptables -t nat -A POSTROUTING -o eth3 -j MASQUERADE`. Dieser Befehl maskiert in R1 ausgehende Pakete mit der momentan an *eth3* aktuellen IP-Adresse und versteckt so das lokale Netz nach aussen hin.

Alternativ können auf den Rechnern im LAN statische Routen in das 10er-Netz, mit der IP-Adresse an *eth3* auf R1 als Gateway, angelegt werden. Dies hat jedoch den Nachteil, dass es auf jedem Rechner eingerichtet werden muss und für Rechner ausserhalb des lokalen Netzes die privaten IP-Adressen unerreichbar bleiben.

5.1.1 Host-Konfiguration

Da sowohl während dem Starten als auch dem Beenden eines Szenarios alle aktuellen Netzwerkkonfigurationen des Host verloren gehen, müssen diese, entweder gesondert nach dem Beenden oder für die laufende Simulation selbst, wiederhergestellt werden. Die automatische Wiederherstellung nach dem Beenden kann durch den <physicalif>-Tag (siehe Kapitel XML Syntax und Semantik) geschehen, wodurch nach dem Herunterfahren eines Szenarios durch den Parser die in diesem Tag abgelegte Konfiguration wiederhergestellt wird. Für die laufende Simulation wird die Host-Konfiguration im <host>-Tag abgelegt, wodurch die Konnektivität des Host zu externen Netzen erhalten bleibt, er jedoch automatisch auch ein Teil der Simulation wird. Werden jedoch keine Routen in das Netz der Simulation gelegt, spielt der Host darin auch keine aktive Rolle. Alternativ kann nach dem Beenden durch den Befehl `rcnetwork restart` (Suse) die Netzwerk-Konfiguration des Host neu initialisiert werden.

In der in 5.1 vorgestellten Konfiguration geht die Konnektivität des Host zum externen Netz verloren, da die Schnittstelle *eth0* ihre IP-Adresse verliert. Da die Bridge im Host jedoch als Schnittstelle repräsentiert wird (siehe Kapitel 2.4.1), und zwar mit der MAC-Adresse der externen Schnittstelle, ist es möglich die Bridge selbst als Schnittstelle zum externen Netz zu verwenden. Gibt man dieser nun eine IP-Adresse, kann z.B. eine Internetverbindung für den Benutzer am Hostrechner aufrecht erhalten werden, obwohl die Simulation ebenfalls die Schnittstelle *eth0* benutzt. Die folgenden Zeilen konfigurieren den Host entsprechend und legen eine default-Route über die neue Schnittstelle *extern* zum Gateway:

```
<hostif net="extern">
 <ipv4 mask="255.255.248.0">192.168.2.102</ipv4>
</hostif>
<route type="inet" gw="192.168.2.1">default</route>
```

Der Host ist entsprechend der Beschreibung jetzt auch Teil des Szenarios, da aber keine Routen in die Netze der Simulation angegeben wurden, bleibt seine Rolle identisch zu der vor dem Start der Simulation. Die vollständige Host-Konfiguration innerhalb des <host>-Tag hat jedoch den Nebeneffekt, dass sich für den Benutzer am Host nichts ändert und weiter volle Konnektivität zum lokalen Netz bzw. Internet besteht. Die Ausgabe von `route` bestätigt, dass nun das LAN (192.168.2.0/24) bzw. das Standardgateway (192.168.2.1) über die neue Schnittstelle *extern* zu erreichen sind:

```
# route -n
Kernel IP Routentabelle
Ziel            Router          Genmask         Flags  Iface
192.168.0.16    0.0.0.0         255.255.255.252 U      R1-ethX
192.168.0.20    0.0.0.0         255.255.255.252 U      R2-ethX
192.168.0.8     0.0.0.0         255.255.255.252 U      VM3-ethX
192.168.0.12    0.0.0.0         255.255.255.252 U      VM4-ethX
192.168.0.0     0.0.0.0         255.255.255.252 U      VM1-ethX
192.168.0.4     0.0.0.0         255.255.255.252 U      VM2-ethX
192.168.2.0     0.0.0.0         255.255.255.0   U      extern
127.0.0.0       0.0.0.0         255.0.0.0       U      lo
0.0.0.0         192.168.2.1     0.0.0.0         UG     extern
```

Die Topologie entspricht jetzt diesem Netz:

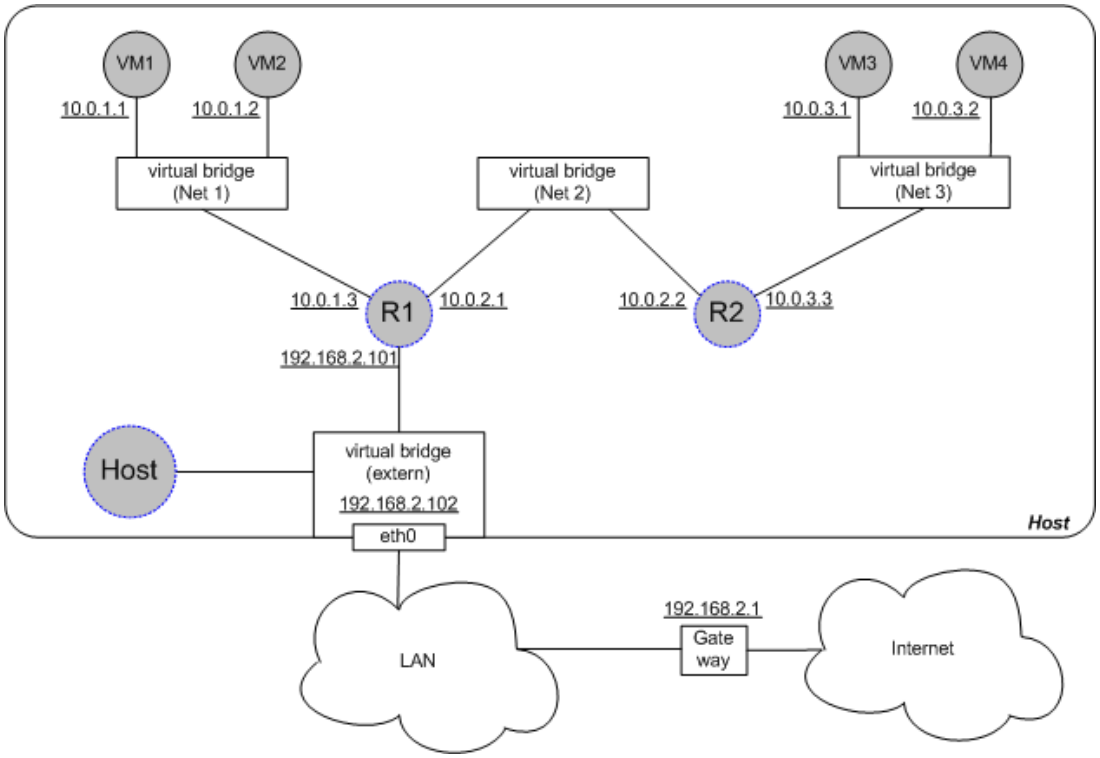

Abbildung 3: Beispielszenario externlayer2_mitHost.xml (XML Beschreibung siehe Anhang B)

5.2 Anbindung auf Layer3

Die zweite Möglichkeit ein Szenario an ein externes Netz anzubinden, besteht in der Nutzung des Hostrechners als Router. Diese Methode ist einfacher umzusetzen und daher besser geeignet, um z.B. via Online-Update neue Software im Filessystem zu installieren, hat jedoch den Nachteil, dass für IP-Pakete ein zusätzlicher Hop entsteht. Damit ist diese Methode nicht geeignet für verteilte Simulationen, da sich die Netztopologie im Übergang zur verteilten Topologie ändert (zwischen zwei Netzknoten würden zwei zusätzliche Hops durch die Hostrechner hinzukommen).

Die physikalische Schnittstellen-Konfiguration des Host wird mit dieser Methode nicht verändert, er erhält lediglich eine zusätzliche virtuelle Schnittstelle die ihn in die Simulation verbindet. Zu diesem Zweck wird ein zusätzliches virtuelles Netz (Net0) deklariert:

```
<net name="Net0"/>
```

Dieses benötigt keine externe Verbindung, da es nur eine „innere“ Verbindung darstellt. Der Host bleibt unverändert an das externe Netz verbunden und routet Verkehr zwischen externem und internem (virtuellem) Netz. Um die Anbindung an das zusätzliche Netz Net0 zu vervollständigen, müssen auf R1 und auf dem Host neue Schnittstellen angelegt werden:

```
<if id="3" net="Net0">
 <ipv4>10.0.0.2</ipv4>
</if>

...
<hostif net="Net0">
 <ipv4>10.0.0.1</ipv4>
</hostif>
```

Um Verkehr zwischen R1 und dem Host routen zu können, muss auf R1 eine statische (default-) Route zum Host gelegt werden:

```
<route type="inet" gw="10.0.0.1">default</route>
```

Auf dem Host muss eine statische Route in das Netz der Simulation (10.0.0.0/16) gelegt und IP-forwarding aktiviert werden. Die vollständige Hostkonfiguration lautet:

```
<host>
 <hostif net="Net0">
  <ipv4 mask="255.255.255.0">10.0.0.1</ipv4>
 </hostif>
 <route type="inet" gw="10.0.0.2">10.0.0.0/16</route>
 <forwarding type="ipv4"/>
</host>
```

Das aktuelle Szenario lässt sich folgendermaßen darstellen:

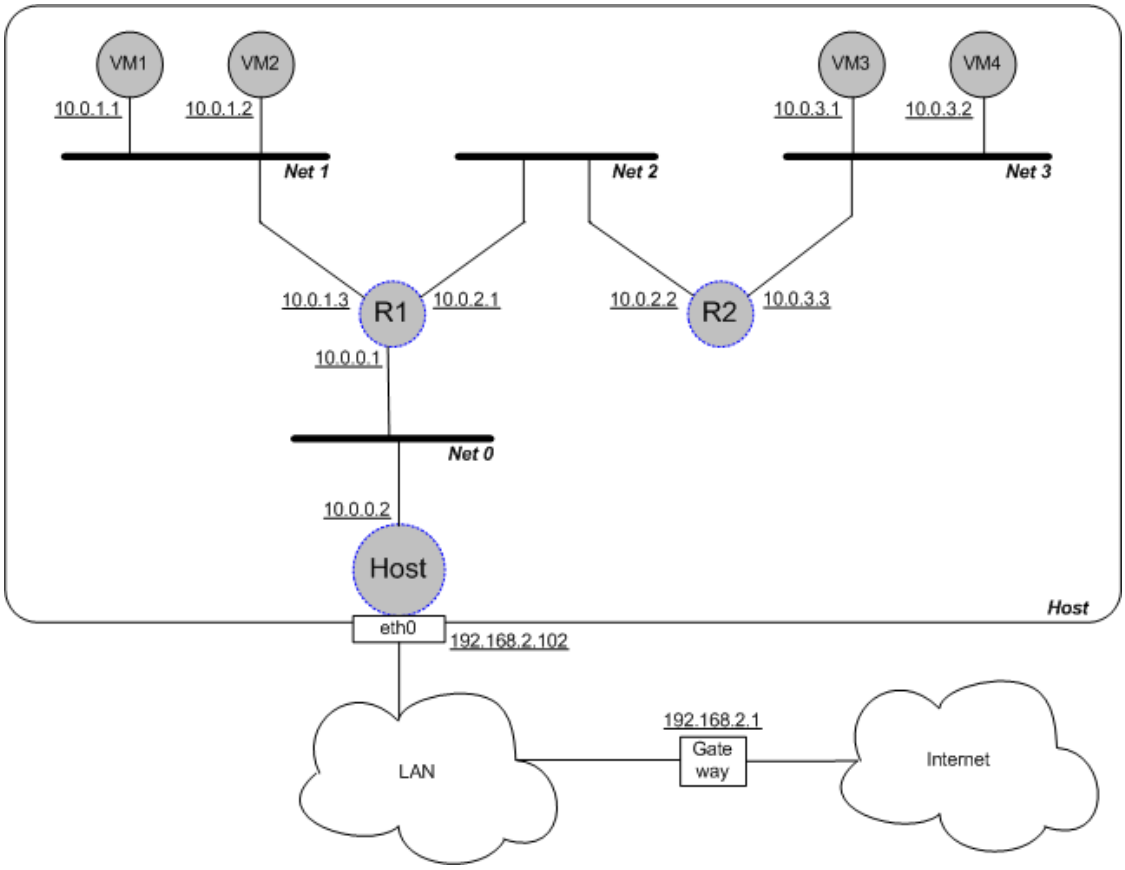

Abbildung 4: Beispielszenario externlayer3.xml (XML Beschreibung siehe Anhang C)

Damit Verkehr von aussen in die Simulation kommen kann, muss auf dem Host NAT bzw. IP-Masquerading (siehe Kapitel 5.1) aktviert werden: `iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE`. Dieser Befehl maskiert alle ausgehenden Pakete mit der momentan an *eth0* aktuellen IP-Adresse und versteckt so das lokale Netz nach aussen hin.
Ein Ping von VM1 an google.de bestätigt die vollständige Erreicharkeit:

```
VM1:~# ping 216.239.39.104
PING 216.239.39.104 (216.239.39.104) 56(84) bytes of data.
64 bytes from 216.239.39.104: icmp_seq=1 ttl=241 time=172 ms
64 bytes from 216.239.39.104: icmp_seq=2 ttl=241 time=208 ms
64 bytes from 216.239.39.104: icmp_seq=3 ttl=241 time=159 ms
64 bytes from 216.239.39.104: icmp_seq=4 ttl=241 time=161 ms

--- 216.239.39.104 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3040ms
rtt min/avg/max/mdev = 159.884/175.626/208.838/19.826 ms
```

Anmerkung: Im Netz der Uni-Koblenz funktioniert die Erreichbarkeitsprüfung nicht, da Icmp-Pakete gefiltert werden.

6 Verteilte Simulationen

6.1 Modellierung

6.1.1 Vorraussetzungen

Vorraussetzung für den Betrieb verteilter Simulationen ist ein Pool von Linux-Rechnern, die über je eine Netzwerkkarte verfügen und über ein Ethernet (z.B. Switch) verbunden sind.
Zudem muss das Bridge-Modul installiert sein (Teil der VNUML-Standardinstallation) und virtuelle Netze müssen mit dem Transporttyp „virtual_bridge“ deklariert sein, da über den Transporttyp „uml_switch“ keine externen Verbindungen möglich sind (siehe Kapitel 2.4). Da die Hostkonfiguration durch die Bridge verändert wird, sind root-Rechte erforderlich.

6.1.2 Placement

Grundlegend für das Arbeiten mit verteilten Simulationen ist die Aufteilung einer Gesamttopologie in Teilszenarios (Placement), die auf verschiedenen Hostrechnern ausgeführt werden können und so das Gesamtszenario bilden. Für diese Aufteilung ist neben der verfügbaren Hardware (siehe Kapitel 4) auch eine sinnvolle Gruppierung anhand der Netzstruktur ausschlaggebend. Besteht das Netz z.B. aus mehreren Autonomen Systemen, so ist eine sinnvolle Gruppierung: 1 bis n AS pro Hostrechner, entsprechend der Größe des AS und der Hardware des Host zugeteilt, da in einer großen Topologie zwischen den AS im Allgemeinen weniger Netze existieren (i.d.R. 1 Netz zur Verbindung von 2 BGP-Routern) als innerhalb eines AS, in dem viele kleinere Netze existieren.
Alle Netze die „zwischen“ den Teilszenarios liegen (also z.B. Netze die Netzknoten X auf HostrechnerA mit Netzknoten Y auf HostrechnerB verbinden), liegen nach Anbindung an die externBridge im selben physikalischen Ethernet. Gewollte Unterteilungen in getrennte lokale Teilnetze gehen somit verloren. Dies kann auf Layer3 jedoch mit der Eigenschaft des IP-Protokolls, Netze über die Netzmaske in logische Subnetze unterteilen zu können, behoben werden.
Ein Subnetz entsteht durch die Unterteilung aller möglichen IP-Adressen in Teilnetze. Die logische Unterteilung des Netzes in Subnetze entspricht meist der physischen Unterteilung in lokale Teilnetze, kann jedoch auch innerhalb eines Netzes erfolgen.[9] Mit Hilfe dieser Eigenschaft können Netze, die auf Layer2 zusammenfallen, auf Layer3 logisch getrennt werden. Zwei Teilszenarios auf 2 Hostrechnern, wobei jede VM mit einer VM auf dem anderen Hostrechner verbunden ist, sind über die Bridge folgendermaßen verbunden:

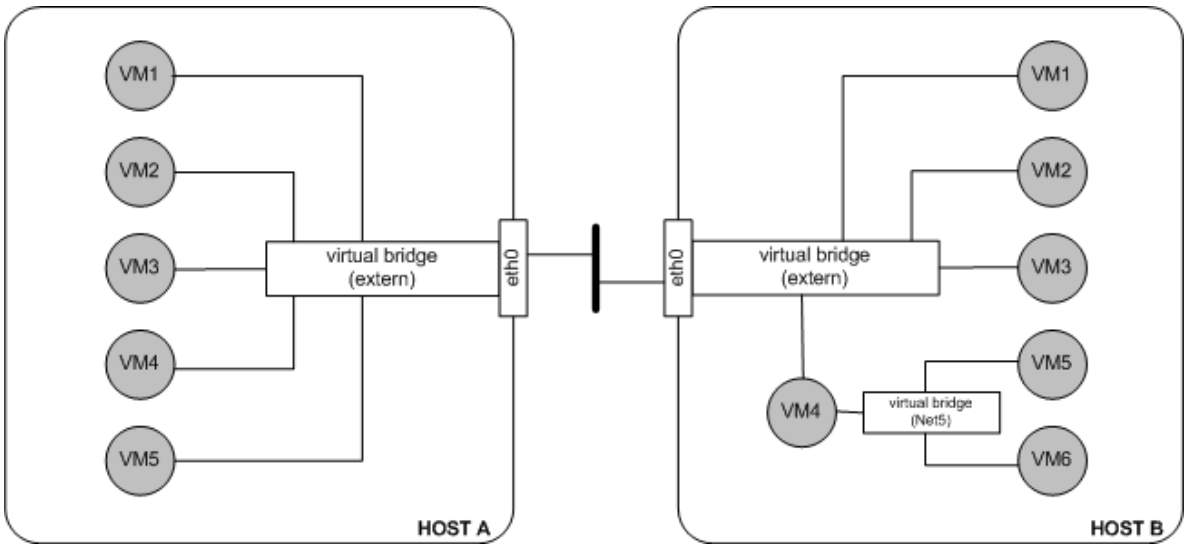

Abbildung 5: Darstellung eines Beispielszenarios auf Layer2

Wird jeder virtuellen Maschine eine IP-Adresse aus einem KlasseC-Netz (mit entsprechender 255.255.255.0 Subnetzmaske) zugewiesen, sind die Teile des Netzes logisch getrennt:

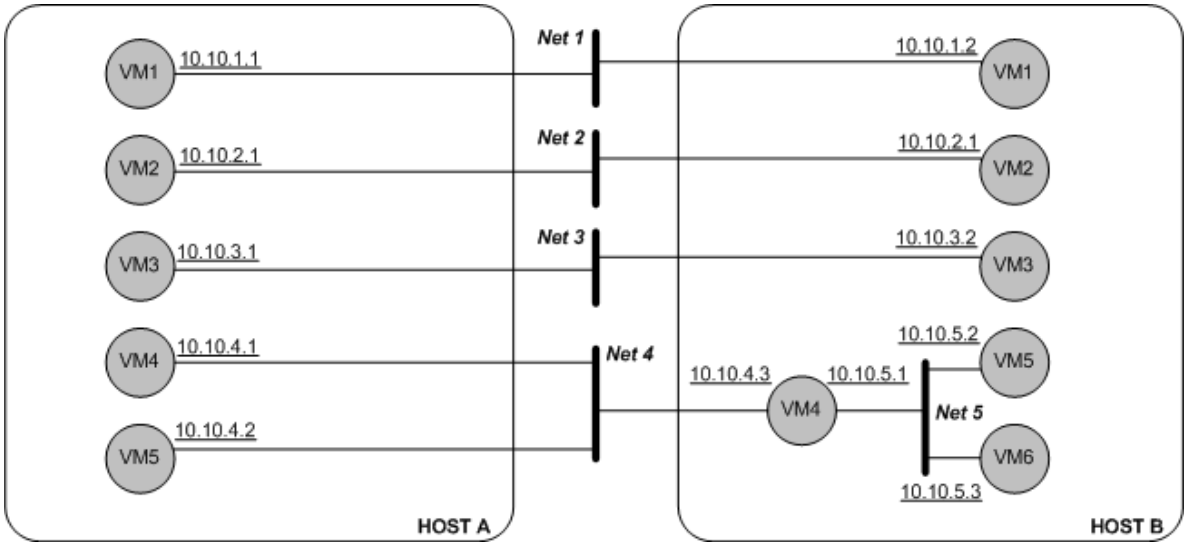

Abbildung 6: Beispielszenarios multia.xml und multib.xml (XML Beschreibung auf der CD)

Ein `ping` von VM1 auf HostA erreicht also nur VM1 auf HostB und keine anderen virtuellen Maschinen auf diesen. Dies gilt analog für VM2 und VM3. VM4 kann VM5 auf HostA und VM4 auf HostB erreichen. Eine Subnetzmaske 255.255.0.0 für VM1 auf HostB und 255.255.252.0 für VM3 auf HostB würde hier z.B. nicht das gewünschte Ergebnis liefern, da VM1 und VM3 dann durch die Überschneidung im gleichen Subnetz liegen, obwohl nach Netz-

topologie keine Verbindung zwischen diesen beiden existieren soll.
Ergebnis dieser Überlegungen ist also, dass ein beliebiges Netz grundsätzlich beliebig aufgeteilt werden kann, da auf IP-Ebene die Topologie logisch erhalten werden kann. Aus dem folgenden Beispiel können n beliebige Knoten auf m beliebigen Hostrechnern ausgeführt werden:

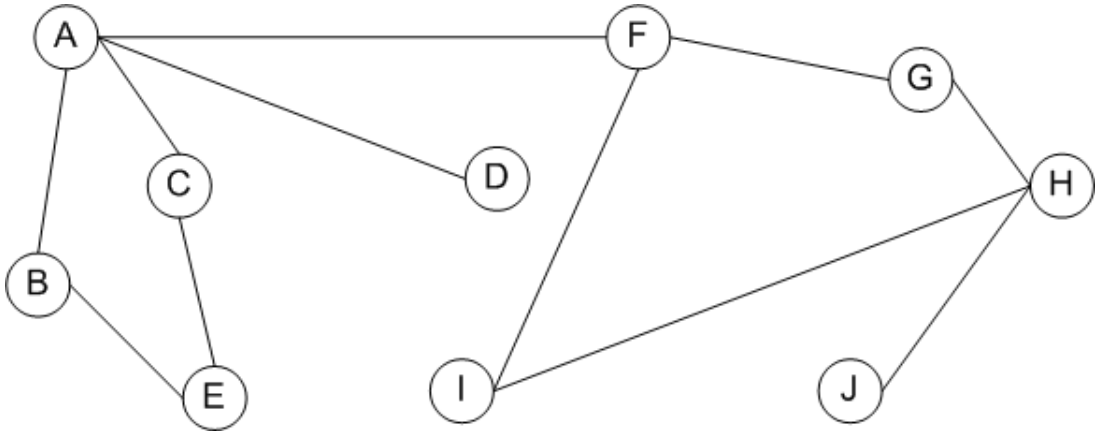

Abbildung 7: beliebige Netztopologie

Die Netzknoten A und J können z.B. auf einem HostrechnerA und die übrigen B,C,D,E,F,G,H,I auf einem HostrechnerB simuliert werden. Dazu müssen die Schnittstellen von A und J und die entsprechenden von B,C,D,F und H auf das extern-Netz gelegt werden und dort über passende Subnetze logisch unterscheidbar gemacht werden. Eine „sinnvolle Aufteilung“ spiegelt jedoch eher zusammengehörige Gruppen des Netzes wieder, da solche Aufteilungen insbesondere für nachfolgende Benutzer intuitiver und besser nachvollziehbar sind. Eine „bessere“ Aufteilgung ist demnach: A,B,C,D,E auf HostrechnerA und F,G,H,I,J auf HostrechnerB. Damit werden möglichst wenig Netze auf das extern-Netz zusammengefasst (nur die Verbindung A-F), was die Struktur des Netzes in einem höheren Grade erhält als die rein logische Unterteilung in Subnetze. Zusätzlich wird der Traffic auf dem real vorhandenen LAN zwischen den Hostrechnern reduziert.

6.1.3 Arbeitszyklus

Der in [2] beschriebene VNUML-Arbeitszyklus unterteilt das Arbeiten mit VNUML in drei Phasen: **Designphase, Implementierungsphase und Ausführungsphase**. Dieser Zyklus bleibt für verteilte Simulationen identisch, seine Umsetzung wird im Folgenden darauf angepasst.

Im ersten Schritt erfolgt die **Designphase**. Diese Phase wird identisch durchgeführt, d.h. das

gewünschte Netz mit allen Eigenschaften und Knotentypen wird als Gesamtszenario geplant und entworfen. Ein Gesamtentwurf des Netzes sollte immer vorhanden sein, da das Netz, obwohl es auf verschiedenen Hostrechnern ausgeführt wird, aus Sicht der virtuellen Netzknoten eine Einheit ohne Unterteilung bildet.
Zusätzlich muss in dieser Phase das „Placement“, also die Aufteilung der Topologie auf die vorhandene Hardware, durchgeführt werden. Dabei müssen die verfügbaren Ressourcen berücksichtigt werden (siehe Kapitel 4). Da die Anzahl der virtuellen Maschinen den entscheidenden Faktor stellt, sollte anhand dieser eine sinnvolle Aufteilung und Gruppierung erfolgen.
Die Verbindungen zwischen den Teilszenarios müssen schon in der Designphase passend gewählt werden, d.h. geeignete Vergabe der IP-Adressen und dazugehörigen Masken für Subnetze die auf dem extern-Netz liegen, um Zusammengehörigkeiten anhand der Subnetzmasken unterscheidbar zu machen (siehe Kapitel 6.1.2). Da VNUML standardmäßig KlasseC-Masken (sofern nicht anders deklariert) für alle virtuellen Schnittstellen vergibt, reicht es aus, IP-Adressen aus verschiedenen KlasseC-Netzen zu vergeben: auf HostA VM1 10.10.1.1, VM2 10.10.2.1 und auf HostB VM1 10.10.1.2, VM2 10.10.2.2.

Im zweiten Schritt folgt die **Implementierungsphase**, in der die XML-Beschreibungen, der in Schritt 1 entworfenen Topologie, erstellt werden. Die Implementierungsphase wird bei n Hostrechnern n-mal durchgeführt, d.h. für jeden Hostrechner wird unabhängig von den anderen das Szenario erstellt, welches ihm entsprechend dem „Placement“ aus Phase 1 zugedacht wurde. Jedes Teilnetz wird dabei in üblicher Weise erstellt (XML-Datei + Konfigurationen der Routingdaemons anlegen), folgende Maßnahmen sind zusätzlich zu beachten:

- Um die MAC-Adressen netzweit eindeutig zu halten, muss in jeder XML-Datei ein passender offset im <automac>-Tag deklariert werden (z.B. auf HostA: `<automac offset='10'/>` und auf HostB: `<automac offset='20'/>`).
- Für alle Netze, die auf der externen Verbindung liegen sollen, wird je XML-Datei ein entsprechendes Netz deklariert: `<net name='extern' external='eth0'/>`.
- Alle Schnittstellen der virtuellen Maschinen, die zum extern-Netz gehören, werden entsprechend deklariert:

```
<vm name="R1">
...
 <if id="2" net="extern">
  <ipv4>10.10.3.1</ipv4>
 </if>
...
```

- Optional: Hostkonfiguration je XML-Datei anpassen (siehe Kapitel 5.1.1), um mit dem Hostrechner -unabhängig von der Simulation- erreichbar zu bleiben (z.B. für eine Internetverbindung oder zum Einloggen per ssh auf den Hostrechner). Die Schnittstellenkonfiguration kann zusätzlich im <physicalif>-Tag für die spätere automatische Wiederherstellung abgelegt werden.

Im dritten Schritt folgt die **Ausführungsphase**, in der die in Phase 1 und 2 erstellten Teilszenarios gestartet werden und ein Gesamtszenario bilden. Ergebnis der Phasen 1 und 2 sind n Szenariobeschreibungen (XML-Dateien + Routerkonfigurationen). Auf jedem der n Hostrechner wird das entsprechende Teilszenario in üblicher Weise hochgefahren: `vnumlparser.pl -t Teilnetz_auf_Host_X.xml -vB`. Analog werden die Daemons in diesem Teilnetz gestartet: `vnumlparser.pl -t start@Teilnetz_auf_Host_X.xml -v`.
Der VNUML-Parser führt dabei alle nötigen Schritte und Konfigurationen selbstständig und transparent durch. Für einen Benutzer, eingeloggt auf einer beliebigen virtuellen Maschine (also einem beliebigen Netzknoten: Host,Router etc), erscheint nach erfolgreichem Start, und ggfs. benötigter Konvergenzzeit, die Gesamttopologie, wie während der Designphase als Gesamtszenario erstellt.

6.2 Umwandlung

In diesem Kapitel wird anhand eines Beispiels der Übergang von einem existierenden Gesamtszenario zum verteilten Szenario demonstriert. Die Topologie selbst soll dabei identisch bleiben, nur die Ausführung wird physikalisch auf mehrere Teilnetze aufgeteilt. Ausgangspunkt ist das folgende Netz, dass in dieser Form komplett auf einem Hostrechner A startet:

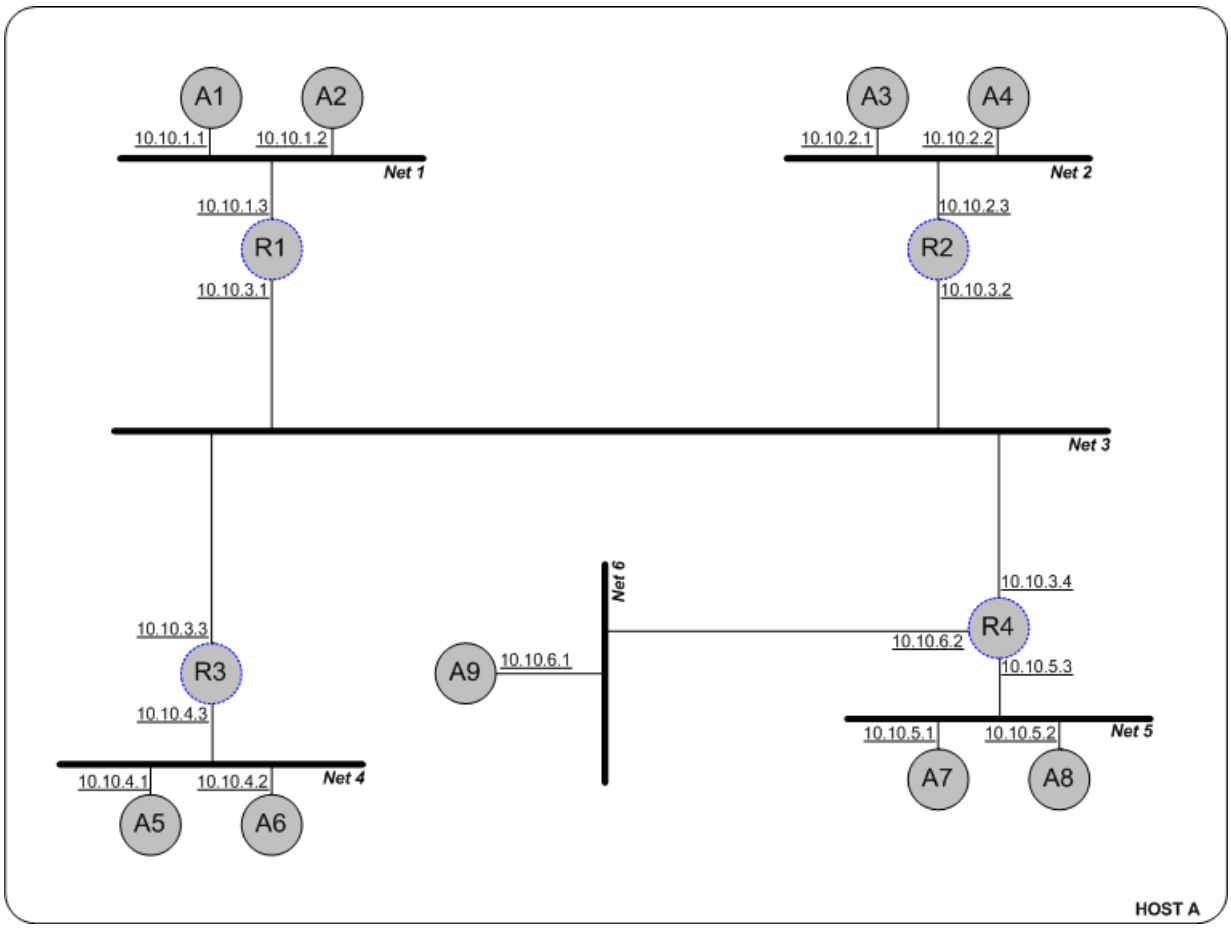

Abbildung 8: Ausgangstopologie - 1 Hostrechner A

R1,R2,R3 und R4 sind über statische Routen als Router für A1 - A9 konfiguriert.
Im ersten Schritt soll das obige Netz (verteilteSim1.xml) so aufgeteilt werden, dass es auf zwei Hostrechnern A und B ausgeführt werden kann, die zusammen die oben dargestellte Topologie erzeugen. Dafür müssen zwei XML-Dateien erstellt werden, welche je ein Teilnetz des Gesamtnetzes enthalten. Die XML-Datei auf dem Hostrechner A (verteilteSim2a.xml) enthält alle virtuellen Maschinen und Netze die im folgenden Bild dem Host A zugeordnet sind. Die XML-Datei auf dem Hostrechner B enthält analog alle dem Host B zugeordneten Elemente:

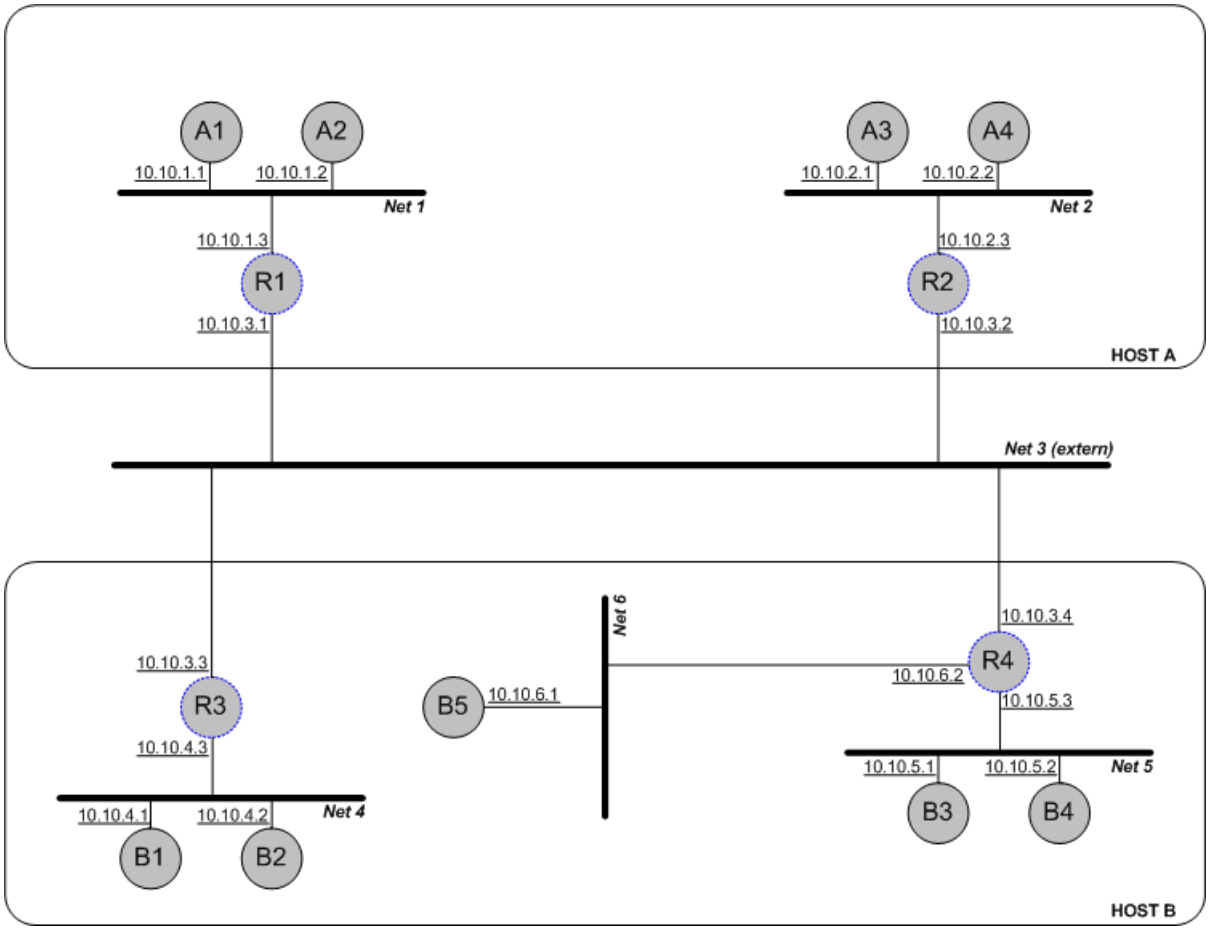

Abbildung 9: Topologie - 2 Hostrechner A,B

Alle Elemente, abgesehen von den Netzen die eine externe Verbindung benötigen, können via copy-and-paste aus dem Ausgangsnetz übernommen werden. Ergebnis sind zwei XML-Dateien die unabhängig voneinander die Teilszenarien für HostA (verteilteSim2a.xml) und HostB (verteilteSim2b.xml) enthalten. Folgende Anpassungen müssen vorgenommen werden, um während der gleichzeitigen Ausführung der beiden Szenarios, auf zwei -über einen Switch verbundenen- Hostrechnern, das gewünschte Gesamtszenario zu erhalten:

- Name der XML-Datei passend wählen und im <simulation_name>-Tag deklarieren.
- Automac mit passendem offset deklarieren, z.B. auf HostA: `<automac offset='10'/>` und auf HostB: `<automac offset='20'/>`.
- Netzdeklarationen die keine internen Verbindungen sind müssen entfernt werden, in diesem Fall die Zeile die Net3 deklariert (`<net name='Net3'/>`). Für alle Netze, die auf der externen Verbindung liegen sollen, wird ein neues Netz deklariert: `<net name='extern' external='eth0'/>`. Der Abschnitt der Netz-Deklaration auf HostA (verteilteSim2a.xml) lautet dann:

```
<net name="Net1"/>
<net name="Net2"/>
<net name="extern" external="eth0"/>
```

und entsprechend auf HostB (verteilteSim2b.xml):

```
<net name="extern" external="eth0"/>
<net name="Net4"/>
<net name="Net5"/>
<net name="Net6"/>
```

- Alle Schnittstellen der virtuellen Maschinen, die zum neuen externen Netz gehören, anpassen. Hier: in verteilteSim2a.xml *eth2* für R1 und R2, und in verteilteSim2b.xml *eth2* für R3 und R4 anpassen, indem die Schnittstelle dem neuen Netz zugewiesen wird:

```
<vm name="R1">
...
 <if id="2" net="extern">
  <ipv4>10.10.3.1</ipv4>
 </if>
...

<vm name="R2">
...
 <if id="2" net="extern">
  <ipv4>10.10.3.2</ipv4>
 </if>
...
```

  Für R3 und R4 analog. Insbesondere ist in diesem Schritt auf die richtige Zuweisung der IP-Adressen und Netzwerkmasken zu achten (siehe Kapitel 6.1.2). Alle Schnittstellen des extern-Netzes liegen auf Layer2 im selben physikalischen Netz und müssen anhand der Subnetzmaske auf Layer3 in verschiedene logische Teilnetze getrennt werden, um ungewollte Erreichbarkeiten zwischen (laut Netzentwurf garnicht verbundenen) Netzknoten zu vermeiden.

- Optional: Hostkonfiguration je XML-Datei anpassen (siehe Kapitel 5.1.1) oder die Schnittstellenkonfiguration im <physicalif>-Tag für die spätere automatische Wiederherstellung speichern.

Im zweiten Schritt soll das Gesamtszenario in vier Teilszenarios aufgeteilt werden:

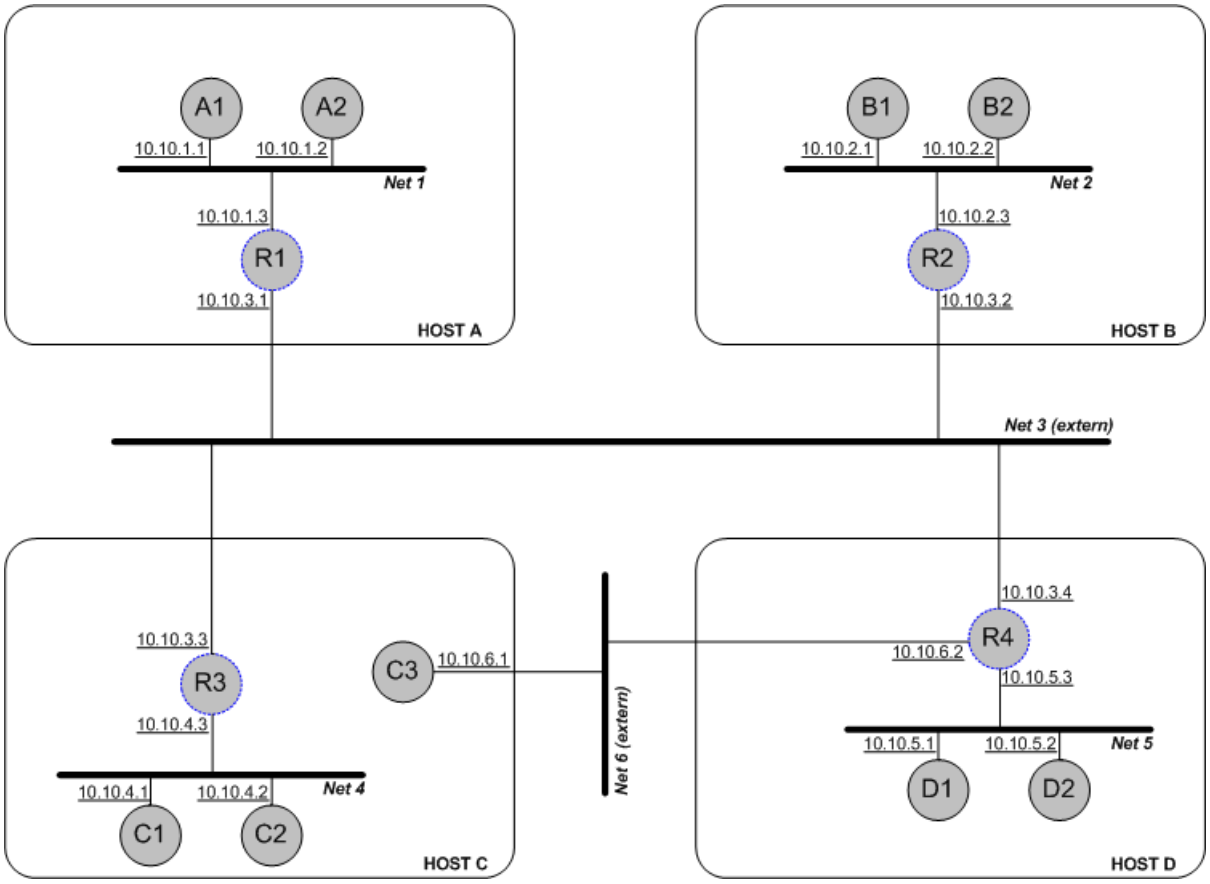

Abbildung 10: Topologie - 4 Hostrechner A,B,C,D

Damit enthält die XML-Datei auf HostA (verteilteSim3a.xml) nur noch die Elemente Net1 und A1,A2,R1. HostB (verteilteSim3b.xml) simuliert Net2 und B1,B2,R2. HostC (verteilte-Sim3c.xml) simuliert Net4 und C1,C2,R3 und C3. HostD (verteilteSim3d.xml) simuliert Net5 und D1,D2,R4. Die vier XML-Dateien können wieder durch copy-and-paste aus der ursprünglichen Datei erstellt werden, da sich die Topologie und die Eigenschaften jeder einzelnen VM nicht ändern und nicht ändern sollen. Die Anpassungen funktionieren analog zu den oben vorgestellen: in jeder der vier XML-Dateien einen offset für die automatische MAC-Adressvergabe und ein Netz mit „external“-Attribut deklarieren und die dazugehörigen Schnittstellen der virtuellen Maschinen anpassen. Die Namen der VM müssen nicht zwingend verändert werden. In diesem Beispiel sollen die neu vergebenen Namen die Zugehörigkeit zu den Hostrechnern veranschaulichen.

Nach einem Testlauf ist zu erkennen, dass diese Unterteilung problemlos funktioniert, obwohl C3 nun nicht mehr direkt mit dem Router R4 verbunden ist. C3 liegt zwar mit R1,R2,R3 und R4 im selben Ethernet, durch die Unterteilung in unterschiedliche Subnetze ist jedoch eine getrennte logische Verbindung auf Layer3 vorhanden. Da C3 im KlasseB-Netz 10.10.6.0/24 und die Router R1 bis R4 im KlasseB-Netz 10.10.3.0/24 liegen, ist auf IP-Ebene eine logische Unterteilung in getrennte Subnetze möglich (siehe Kapitel 6.1.2). Die folgende Abbildung zeigt die Umsetzung der Ethernet-Topologie mit Hilfe der „virtual_bridge“:

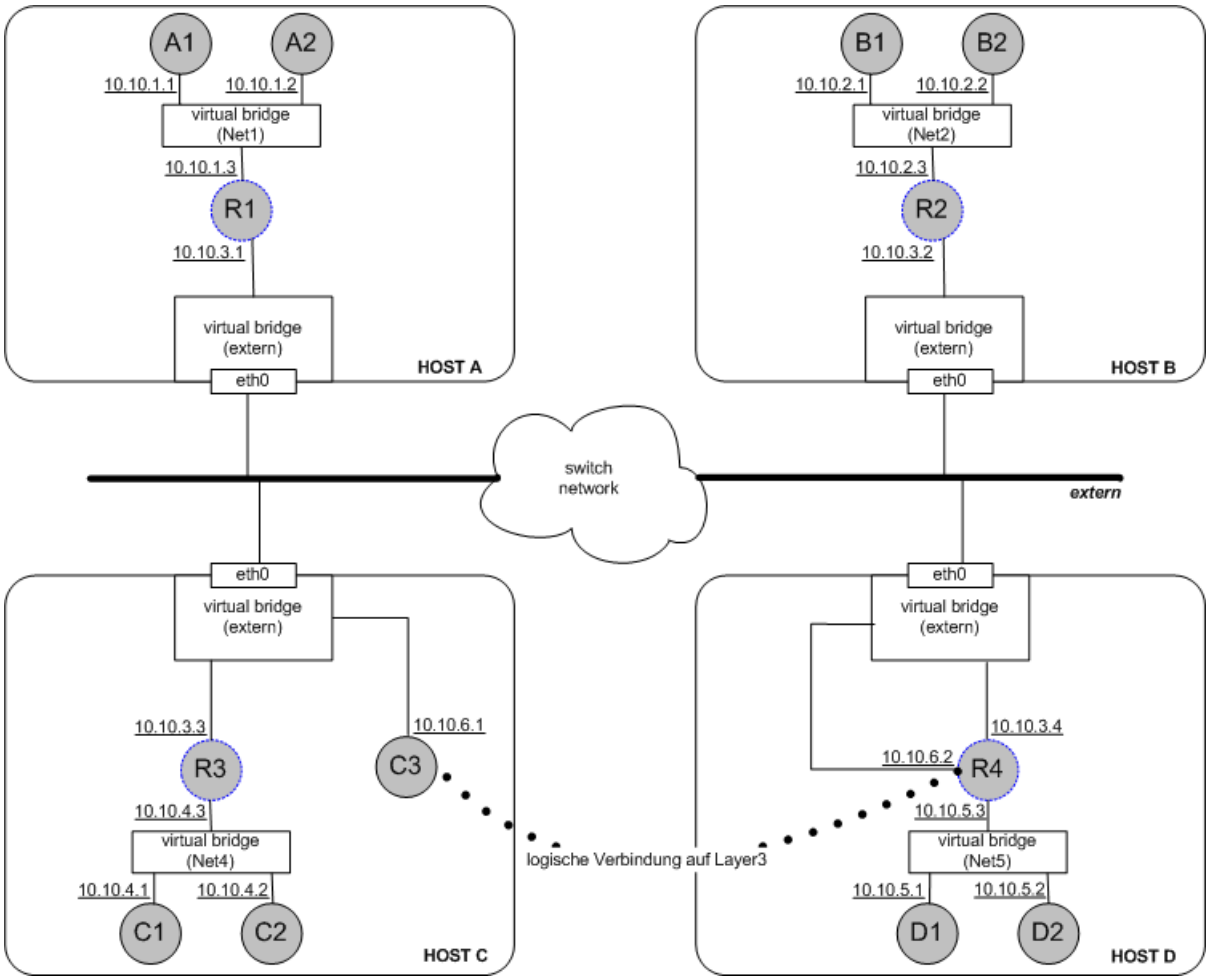

Abbildung 11: Darstellung auf Layer2

Die vollständigen XML-Beschreibungen befinden sich auf der beiliegenden CD.

6.3 Beispielnetz bignet

Auf Grundlage der, in den letzten beiden Kapiteln, vorgestellten Prinzipien soll nun exemplarisch ein Netz erstellt werden, dass aufgrund seiner Größe nicht mit normalem Aufwand auf einem einzigen Hostsystem zu starten ist.

Die Abbildung im Anhang D zeigt die Topologie des Netzes. Um der realen Netzwelt möglichst nahe zu kommen, werden in diesem Beispiel Netzknoten bewusst in Router und Host unterschieden (Host entspricht hier einer virtuellen Maschine, die keine aktive Routing-Aufgabe im simulierten Netz übernimmt). Diese Unterscheidung ermöglicht die Prüfung von Erreichbarkeiten zwischen zwei solchen Host, im Gegensatz zur reinen Router-Simulation, in der nur Router simuliert werden deren propagierte Netze garnicht existieren. Deshalb hängt in dieser Topologie an jedem Router mindestens ein Klasse B Netz, in dem exemplarisch ein Host vertreten ist.
Das Netz besteht aus 6 Autonomen Systemen, wobei in jedem AS ein BGP-Router die globalen Routing-Aufgaben übernimmt. Die Autonomen Systeme AS65001 - AS65006 seien ab sofort mit AS1 bis AS6 bezeichnet. AS1 kann als großer ISP angesehen werden, da er das Null-Prefix bekannt macht und so jeder BGP-Router eine default-Route zu R1 hat. Zusätzlich geben alle BGP-Router ihre direkt angebundenen Netze an ihre Nachbarn bekannt (*redistribute connected*). Auf den Routern R5 und R6 läuft nach aussen BGP und auf den beiden inneren Schnittstellen RIP. Die vom RIP Prozess gelernten Routen des jeweils eigenen AS geben R5 und R6 zusätzlich an ihre BGP-Nachbarn bekannt (*redistribute rip*). Dies verdeutlicht den raschen Anstieg der Größe der Routing-Tabellen der BGP-Router (>50 Einträge), da R1,R2,R3 und R4 neben den aggregierten Prefixen auch sämtliche spezifischen Prefixe aus AS5 und AS6 kennen. R4 hingegen propagiert nur den aggregierten Prefix (164.0.0.0/8) für AS4. Über eine statische default-Route schickt er Verkehr in das eigene AS an den Backbone-Router der OSPF-Area0 (AS4R1). Dieser schickt entsprechend für andere AS bestimmten Verkehr über eine statische default-Route zum BGP-Router R4.
Innerhalb von AS4 läuft OSPFv2, aufgeteilt in die Area0 (Backbone Area), Area1 und Area2. Innerhalb von AS5 und AS6 läuft RIPv2. Um Komplexität zu vermeiden wird auf Zyklen innerhalb der RIP und BGP Netze verzichtet und Verkehr für unbekannte Zielnetze wird innerhalb der AS über statische Routen zu den jeweiligen BGP-Routern geleitet.

Eine sinnvolle Aufteilung des Netzes für eine verteilte Simulation kann durch Gruppierung der Autonomen Systeme erfolgen. AS1,AS2 und AS3 sollen aufgrund ihrer Größe zusammen auf einem Hostrechner ausgeführt werden. AS4, AS5 und AS6 hingegen jeweils auf einzelnen Rechnern. Die Abbildung im Anhang E verdeutlicht diese Aufteilung. Die Subnetze die auf den Verbindungen zwischen den 6 BGP-Routern liegen überschneiden sich nicht, weshalb die Verbindungen ohne Änderungen auf das externe Netz gelegt werden können. Die Verbindun-

gen von R1 zu R3, R1 zu R2 und R2 zu R3 befinden sich weiterhin in der Simulation eines Hostrechners und können unverändert bleiben. In der XML-Beschreibung jedes Teilszenarios werden nun die restlichen Inter-AS-Verbindungen konfiguriert, in dem die jeweils dazugehörigen Schnittstellen zum Netz, welches als *external* deklariert wurde, hinzugefügt werden. Für R1 gilt:

```
...
<!-- Verbindungen zwischen den AS -->
<net name="extern" external="eth0"/>

<vm name="R1">
 <if id="1" net="AS1-AS2">
  <ipv4 mask="255.255.255.0">161.0.2.1</ipv4>
 </if>
 <if id="2" net="AS1-AS3">
  <ipv4 mask="255.255.255.0">161.0.1.5</ipv4>
 </if>
 <if id="3" net="extern">
  <ipv4 mask="255.255.255.0">161.0.3.9</ipv4>
 </if>
 <if id="4" net="AS1Net1">
  <ipv4 mask="255.255.0.0">161.1.0.1</ipv4>
 </if>
 <if id="5" net="AS1Net2">
  <ipv4 mask="255.255.0.0">161.2.0.1</ipv4>
 </if>
<forwarding/>
...
</vm>

...
```

Die Schnittstelle eth3 (161.0.3.9) auf R1 wird also über die extern-Bridge mit dem physikalischen Netz verbunden. Die andere Seite dieser Verbindung zwischen den BGP-Routern R1 und R4 wird analog konfiguriert. Die Schnittstelle eth1 (161.0.3.10) von R4 wird in der XML-Datei des Rechners der AS4 simuliert, folgendermaßen deklariert:

```
...
<!-- Verbindungen zwischen den AS -->
<net name="extern" external="eth0"/>
```

```
<vm name="R4">
 <if id="1" net="extern">
  <ipv4 mask="255.255.255.0">161.0.3.10</ipv4>
 </if>
 <if id="2" net="extern">
  <ipv4 mask="255.255.255.0">162.0.2.6</ipv4>
 </if>
 <if id="3" net="Lan01">
  <ipv4 mask="255.255.0.0">164.1.0.1</ipv4>
 </if>
 <if id="4" net="Net12">
  <ipv4 mask="255.255.255.0">164.0.12.1</ipv4>
 </if>
<route type="inet" gw="164.0.12.2">164.0.0.0/8</route>
<forwarding/>
...
</vm>

...
```

Die vollständigen XML-Beschreibungen befinden sich auf der beiliegenden CD. Zum Starten des Gesamtszenarios wird auf jedem Hostrechner das entsprechende Teilszenario mit dem Befehl `vnumlparser.pl -t bignet.xml -vB` hochgefahren. Anschließend wird über den Befehl `vnumlparser.pl -x start@bignet.xml -v` die Simulation gestartet (readme.txt für weitere Informationen beachten). Für den Benutzer, der sich in einer beliebigen VM des Netzes befindet, sieht die Topologie nun aus wie zuvor beschrieben und im Anhang D dargestellt.

7 Fazit und Ausblick

Abschließend lässt sich feststellen, dass der Netzwerksimulator VNUML -mit den bereits mitgelieferten Werkzeugen- gute Mechanismen zum Aufbau externer Verbindungen und verteilter Simulationen bereitstellt. Alle dazu nötigen Konfigurationen werden auf eindeutige Weise über eine einfache XML-Sprache spezifiziert. Die genaue Systemkonfiguration der virtuellen Maschinen und des Host führt der Parser anhand der XML-Beschreibung durch und versteckt somit komplexe Details des User-Mode-Linux Pakets vor dem Benutzer. Durch die Ausführung mehrerer Teilszenarios auf mehreren Hostrechnern, die auf logischer Ebene ein Gesamtszenario bilden, ist es möglich die Schwachstelle 'Hardwareanforderungen' von VNUML zu umgehen und somit auch die Untersuchung größerer Netze zu ermöglichen. Die Größe kann dabei je nach verfügbarer Hardware im Bereich 30 - 100 Netzknoten liegen.

Die Ergebnisse dieser Studienarbeit können als Grundlage für spätere Arbeiten dienen, die sich mit großen VNUML-Topologien beschäftigen bzw. Routing-Szenarios größeren Ausmaßes untersuchen. Denkbar wäre z.B. ein großes „Schulungsszenario für Routingalgorithmen", an dem in aufbauender Weise anhand eines gleichbleibenden Beispielnetzes das Verhalten und Zusammenspiel von mehreren (in Quagga verfügbaren) IGP und EGP Protokollen demonstriert werden kann.

Literaturverzeichnis

[1] Andre Volk, Tim Keupen. „Anleitung zur Installation des Netzwerksimulators VNUML“. Uni Koblenz, Projektpraktikum Routingssimulation, SS2005.

[2] Projekthomepage. „http://jungla.dit.upm.es/˜vnuml/“. Technical University of Madrid (UPM). Spain.

[3] Bridge (Netzwerk) „http://de.wikipedia.org/wiki/Bridge_(Netzwerk)“. Wikipedia.de - Die freie Enzyklopädie.

[4] Linux IP Masquerading HOWTO. David Ranch, Ambrose Au, Johann Maas. Oktober 2000.

[5] User-Mode Linux Einführung. Tim Keupen. November 2005. Seminar User-Mode Linux. Universität Koblenz.

[6] Linux Ethernet bridging „http://bridge.sourceforge.net“.

[7] Gentoo News. Linux Virtualisierungstechniken, 13.12.2004 http://www.gentoo.org/news/de/gwn/20041213-newsletter.xml.

[8] UMLinux als Sandbox. Kerstin Buchacker, Hans-Jörg Höxer und Volkmar Sieh. http://www3.informatik.uni-erlangen.de/ Persons/hshoexer/publications/bsi2003.pdf Deutscher IT-Sicherheitskongreß des BSI, 2003.

[9] Subnetz (Netzwerk) „http://de.wikipedia.org/wiki/Subnetz“. Wikipedia.de - Die freie Enzyklopädie.