Netzwerkexperimente mit NEPI

Tobias Betz
Betreuer: Matthias Wachs
Seminar Future Internet SS2014
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: betz@in.tum.de

KURZFASSUNG

Um das Design, die Implementierung und das Verhalten von Netzwerkprotokollen und Applikationen zu analysieren, greifen Forscher und Entwickler oft auf die Hilfe von Experimenten zurück. Eine Möglichkeit dafür ist die Verwendung von Testbeds. Die verschiedenen Testbeds sind heterogen, was eine Anpassung der Definition eines Experiments bei Benutzung eines neuen Testbed erfordert. Ein einzelnes Netzwerkexperiment ist des weiteren in mehrere Schritte des Deployments, der Steuerung und Analyse aufgeteilt, die für jede Testumgebung anders sind. Dadurch ergibt sich eine hohe Komplexität in der genauen Definition von Netzwerkexperimenten und deren Wiederholbarkeit wird gefährdet.
Dieses Paper gibt eine Einführung in NEPI, einem Framework für Netzwerkexperimente, das die Probleme der Definition von Experimenten und der Vielfalt der zu beachtenden Schritte eines Experiments zu lösen versucht. Hierzu werden die grundlegenden Konzepte von Netzwerkexperimenten betrachtet, wie NEPI diese realisiert, und wie Experimente definiert und ausgeführt werden. Außerdem werden die von NEPI unterstützten Testbeds vorgestellt.

1. EINLEITUNG

Netzwerkexperimente geben Forschern und Entwicklern die Möglichkeit Protokolle und Anwendungen in einer sinnvollen Art zu evaluieren. Durch sie kann das Verhalten analysiert und Probleme bei der Verwendung gefunden werden.
Ein Netzwerkexperiment kann beispielsweise die Skalierbarkeit einer Anwendung nachweisen, oder Fehlverhalten in Umgebungen mit verlustbehafteter Kommunikation aufzeigen. Die Experimente dienen dadurch als Instrument, um neue Ideen zu verwirklichen und vorhandene Konzepte zu validieren.
Es stehen verschiedene Arten von Experimentierumgebungen zur Verfügung. Experimente können auf realen Geräten mit der tatsächlichen Software durchgeführt werden, oder von einem Simulator oder Emulator virtualisiert werden.
Simulatoren verwenden statt der tatsächlichen Software nur Modelle derselben. Dadurch erreichen Simulatoren nicht den gleichen Realitätsgrad wie Emulatoren oder reale Testbeds, weil beispielsweise das benutzte Softwaremodell fehlerhaft sein kann. Andererseits bietet eine Simulation exakte Wiederholbarkeit und gute Kontrolle des Experiments, auch aufgrund der fehlenden Echtzeit-Notwendigkeit.
Die Verwendung von realen Geräten schränkt die Skalierbarkeit von Experimenten stark ein, weil die Verwendung von mehr oder komplexeren Geräten automatisch mit höheren Nutzungskosten verbunden ist. Das Deployment, was der Vorbereitung des Testbeds zur Durchführung des Experiments entspricht, ist bei Simulationen einfacher zu handhaben als bei realen Testbeds.
Emulatoren arbeiten in Echtzeit und instrumentieren die echte Software. Emulatoren stellen eine Art Mischlösung aus Simulatoren und realen Testbeds dar. Sie bieten wegen der Instrumentierung gute Skalierbarkeit, und mehr Realitätsbezug als Simulatoren, sind aber bezüglich Kontrolle und Wiederholbarkeit nicht auf dem gleichen Niveau wie ein Simulator. [1]
In Tabelle 1 werden die verschiedenen Arten von Testumgebungen anhand der genannten Faktoren noch einmal dargestellt.

	Realismus	Kontrolle	Deployment	Wiederholbarkeit
Testbed	+	-	-	-
Emulation	○	○	○	○
Simulation	-	+	+	+

Tabelle 1: Vergleich von physikalischer Umgebung, Emulation und Simulation [2]

Eine Testumgebung kann als reales Testbed oder als Virtualisierung vorhanden sein, wobei für virtuelle Testumgebungen die Technik der Simulation oder Emulation angewendet werden kann. Des weiteren haben konkrete Testumgebungen auch noch andere wichtige Unterschiede und Eigenschaften. So können sich Testumgebungen beispielsweise dadurch unterscheiden, wie viele Geräte für das Experiment zur Verfügung stehen, oder benötigt werden. Das muss aber auch mit den Anforderungen des Experiments verglichen werden.
Außerdem liegt die Zielsetzung von verschiedenen Testbeds in unterschiedlichen Arten von Experimenten. Bei einer Testumgebung deren Hauptgebiet beispielsweise Drahtlose Kommunikationstechnologien sind, macht ein Wireless-Experiment am meisten Sinn. Die möglichen Testumgebungen für ein Experiment sind also sehr heterogen und unterscheiden sich in zentralen Aspekten, wie der inneren hierarchischen Struktur und der damit einhergehenden Art des Zugriffs.
Auch Zugriff auf einzelne Geräte der Testbeds erfolgt unterschiedlich. Ein einfacher Linuxrechner als Teilnehmer einer Testumgebung kann beispielsweise durch das Protokoll SSH gesteuert werden, aber ein verteiltes Netzwerk aus Sensorknoten verlangt nach anderen Arten des Zugriffs.
Der Lebenszyklus eines Netzwerkexperiments läuft in mehreren Schritten ab. Zuerst muss es geplant und definiert werden, um eine einheitliche Darstellung des Experiments vor-

doi: 10.2313/NET-2014-08-1_02

weisen zu können. Hier kann es bereits sein, dass die Eigenheiten der Testbeds miteinbezogen werden müssen. Nach der Definition kann das Experiment ausgeführt werden. Dieser Lebenszyklus kann wieder in Teilschritte unterteilt werden. Zunächst muss der Zugriff auf die teilnehmenden Geräte des Testbeds und deren Konfiguration erfolgen, was dem Deployment entspricht. Anschließend kann das Experiment gestartet und kontrolliert werden. Zum Abschluss werden die Ergebnisse des Netzwerkexperiments evaluiert und in hinsichtlich seiner Fragestellung untersucht. Diese Ergebnisse können als Planungsgrundlage für nachfolgende Experimente dienen.
Durch die erwähnten Unterschiede in den verschiedenen Testbeds, die teilweise sehr umfangreich sind und durch die Anzahl an Schritten, die in einem Experiment notwendig sind, ist die Definition und der gesamte Ablauf eines Netzwerkexperiments für Forscher und Entwickler sehr unübersichtlich.
Es ist sehr aufwendig, bei einer Änderung der Experimentdefinition jeden dieser Aspekte per Hand anpassen zu müssen. Aus diesem Grund ist ein Tool gefragt, das die Komplexität und den Konfigurationsaufwand, der durch die Unterschiede verschiedener Testbeds entsteht, zu minimieren.
In diesem Paper wird mit NEPI[1] ein Werkzeug vorgestellt, das entwickelt wurde um die genannten Probleme zu umgehen. NEPI steht für Network Experiment Programming Interface, und ist ein Framework, das es ermöglicht, auch komplexe Netzwerkexperimente durch ein einzelnes Skript einheitlich zu steuern. Version 3.0 von NEPI wurde im Dezember 2013 veröffentlicht und ist zum aktuellen Zeitpunkt das neueste Release. Auf dieser Version liegt der Fokus in diesem Paper. [3]
In Kapitel 2 wird der grundlegende Aufbau von NEPI erklärt. Anschließend in Kapitel 3 wird der typische Ablauf eines Experiments mit NEPI kurz vorgestellt. In Kapitel 4 wird erläutert, wie ein Experiment in NEPI definiert wird. Danach wird in Kapitel 5 gezeigt, wie die Durchführung von Experimenten funktioniert. In Kapitel 6 werden die von NEPI unterstützten Testbeds und Umgebungen vorgestellt, bevor in Kapitel 8 das Paper noch zusammengefasst wird.

2. STRUKTUR VON NEPI

Wie bereits in der Kapitel 1 erwähnt, ist NEPI ein Framework für Netzwerkexperimente. Es wurde von Planete Team bei INRIA, einer französischen Forschungseinrichtung, entwickelt und ist mittlerweile bei Version 3.0 angekommen. Das nächste Release ist für April/Mai 2014 geplant.
NEPI versucht die Schwierigkeiten, die durch die Unterschiedlichkeiten verschiedener Testumgebungen entstehen, auszugleichen, indem es als einzelnes Werkzeug alle Aspekte zur Durchführung eines Netzwerkexperiments abdeckt. Dadurch ist es möglich, die Konfiguration für Experimente an einem zentralen Punkt zu regeln. NEPI ermöglicht es also, ein Experiment mittels einer einzigen Experimentdefinition, die alle beteiligten Komponenten und Abläufe enthält, festzulegen. Zu den Aufgaben, die NEPI erfüllt, gehört das Deployment, die Kontrolle, die Ausführung und die Ergebnissammlung für ein Netzwerkexperiment. NEPI bietet einen generischen Ansatz für die Ausführung von Experimenten auf verschiedenen Testbeds. Um unterschiedliche Testbeds zu unterstützen, verwendet NEPI ein modulares Prinzip, was die Einbindung noch nicht unterstützter Testbeds ermöglicht. Das zentrale Steuerelement von NEPI heißt *Experiment Controller*. Der Experiment Controller (EC) steuert die *Resource Manager* (RM), die eine gemeinsame Schnittstelle teilen. Ein Resource Manager übernimmt die Steuerung eines Testbed und abstrahiert dessen Funktionen für den Experiment Controller durch Verwendung eines Interface.
In Abbildung 1 ist die Modularität von NEPIs Resource Managern veranschaulicht. Die beiden Resource Manager verwenden hier mit SSH bzw. XMPP unterschiedliche Protokolle zur Steuerung ihrer Testumgebungen.

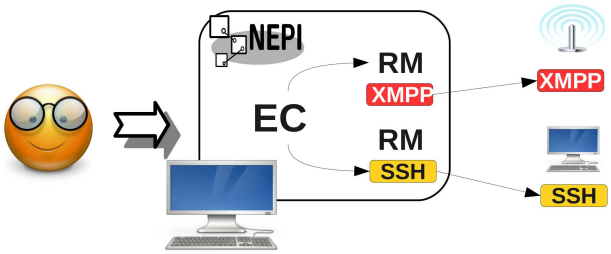

Abbildung 1: Modularität von NEPI [4]

In früheren Versionen von NEPI (v2.0) konnte die Experimentdefinition als "Boxes and Connectors"-Diagramm erfolgen. Dabei werden die beteiligten Geräte und Anwendungen als Boxen dargestellt und ihre topologische Zusammengehörigkeit als Verbindungen zwischen den Boxen. Dieses Diagramm wurde mit einem NEPI-Design-Frontend (NEF) erstellt und konnte direkt verwendet werden. In der aktuellen Version 3.0 ist dieses Feature nicht mehr vorhanden, aber als Veranschaulichung des Experimentdesigns ist das "Boxes and Connectors"-Diagramm trotzdem hilfreich.
NEPI ist als Pythonbibliothek realisiert und ein Benutzer muss nur ein einziges Pythonskript erstellen, das den Ablauf eines gesamten Experiments abdeckt. In diesem Skript wird die Definition des Experiments beschrieben, was bedeutet, dass alle benötigten Ressourcen erzeugt, deren Eigenschaften gesetzt und Abhängigkeiten beschrieben werden. Anschließend kann das Experiment gestartet und Ergebnisse von den teilnehmenden Ressourcen geladen werden. [3]

3. ABLAUF EINES EXPERIMENTS MIT NEPI

Will ein Anwender nun NEPI verwenden, beginnt er mit der Definition des Experiments. Dazu gehört die Beschreibung aller beteiligten Geräte und Abläufe. Diese Definition geschieht in einem Python-Skript, das zur Ausführung des Experiments gestartet wird.
Während der Ausführung kann der Anwender das Experiment kontrollieren, indem er mit der Python-Bibliothek interagiert. Wenn das Netzwerkexperiment beendet ist, ist er in der Lage die gewonnenen Resultate von dem Knoten herunterzuladen, zu analysieren und untersuchen.

[1] `http://nepi.inria.fr/Main/WebHome`

doi: 10.2313/NET-2014-08-1_02

4. EXPERIMENTDEFINITION MIT NEPI

Experimente werden in NEPI mit einem "Boxes and Connectors"-Diagramm definiert und dargestellt. Die Boxen stellen einzelne Ressourcen des Experiments dar, die miteinander verbunden sind.

4.1 Ressourcen

Als *Ressource* kann beispielsweise ein Netzwerkhost, oder eine Netzwerkkarte gesehen werden. Aber auch eine Anwendung, die auf einem Host läuft, eine virtuelle Maschine, oder ein Teil eines Sensornetzes kann eine Ressource repräsentieren. Der Typ einer Ressource ist dann `LinuxHost`, `Interface`, etc.

Jede Ressource hat bestimmte Attribute. Ein Linuxhost hat als Attribute beispielsweise seinen Hostnamen, seine IP-Adresse und den SSH-Port. Für eine Linux-Applikation ist vermerkt, auf welchem Host sie läuft, welche Software dort vorhanden sein muss, und wie der genaue Befehl lautet, der ausgeführt werden soll.

Zur Ergebnissammlung besitzt jede Ressource so genannte *Traces* (engl. Spuren). Bei einer Applikation kann das beispielsweise die Standardausgabe (stdout) und die Fehlerausgabe (stderr) sein, die während des Experiments erzeugt werden.

Die Traces können am Ende des Experiments oder schon während seiner Ausführung vom Benutzer heruntergeladen werden. Sie dienen zur Fehlererkennung, Fehlerbehebung und allgemein zum Debugging. In Abbildung 2 ist die beispielhafte Darstellung einer Linux-Applikations-Ressource gezeigt.

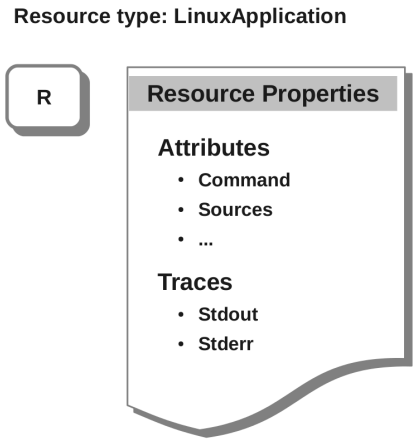

Abbildung 2: Darstellung einer Ressource [3]

Um einen strukturierten Lebenszyklus für Ressourcen zu gewährleisten, hat jede Ressource auch einen Zustand. Dieser Zustand kann beispielsweise angeben, dass die Ressource betriebsbereit ist, noch gefunden werden muss, oder ein Fehler entstanden ist.

Eine Besonderheit von NEPI ist, dass die Zustände der Ressourcen nicht zwingend miteinander synchronisiert sein müssen, d.h. es ist beispielsweise nicht unbedingt nötig, dass eine Ressource zu Experimentbeginn vorhanden ist. [3]

In Abbildung 3 sind die einzelnen Zustände der Ressourcen dargestellt. Der initiale Zustand für alle Ressourcen ist `NEW`, gefolgt von `DISCOVERED`, wenn ein passendes Gerät vom Resource Manager gefunden wurde. Anschließend versucht NEPI die Ressource für den Benutzer verfügbar zu machen, was beispielsweise per SSH-Protokoll passieren kann, und der Zustand wird auf `PROVISIONED` gesetzt. Wenn die Ressource dann konfiguriert wurde, entspricht das dem Zustand `READY`. Eine Ressource kommt in den Zustand `STARTED`, wenn sie erfolgreich gestartet wurde und in den Zustand `STOPPED`, wenn die Ressource für das Experiment nicht mehr benötigt wird.

Wenn der Zustand einer Ressource `FAILED` ist, wurde ein Fehler erkannt, beispielsweise ist die Ressource nicht mehr erreichbar. Der finale Zustand für alle Ressourcen ist `RELEASED`.

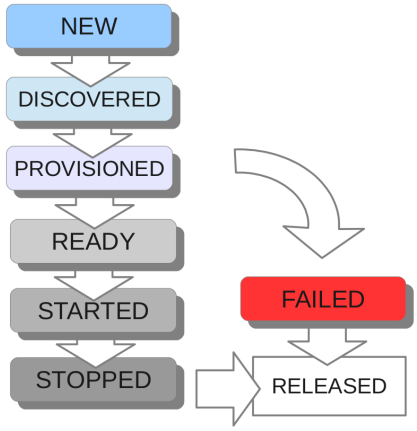

Abbildung 3: Verschiedene Zustände die eine Ressource durchläuft [3]

4.2 Topologie der Ressourcen

Ressourcen hängen oft topologisch voneinander ab. Eine Applikation kann beispielsweise nicht laufen, wenn der dazugehörige Rechner nicht gestartet ist. Das Diagramm, in dem die Ressourcen beinhaltet sind, ist ein topologischer Graph. Darin sind die Ressourcen als Boxen gezeichnet und einige der Ressourcenattribute werden als Verbindungen zwischen den Boxen dargestellt. Eine Linux-Node-Ressource kann als Attribut eine Applikation haben, die auf ihm laufen soll, und die entsprechende Applikations-Ressource hat als Attribut den Linux-Node.

In einem solchen symmetrischen Fall können die Attribute als Verbindungen zwischen Boxen dargestellt werden. Anders ist es bei Attributen wie dem Hostnamen, die nicht als Verbindung zu einer anderen Ressource gesehen werden kann. Diese Attribute sind im "Boxes and Connectors"-Graph nicht zu sehen. Ein weiterer wichtiger Aspekt, der in diesem Graph nicht sichtbar gemacht wird, ist der genaue Ablauf des Experiments.
[3]

In Abbildung 4 wird ein Beispiel eines solchen Graphen gezeigt. Die Detailgenauigkeit ist relativ hoch, weil in diesem Beispiel die beiden Nodes A und B nicht abstrahierend direkt miteinander verbunden sind.

Ein Teilausschnitt des Graphen der die Applikation S und den Host A enthält, ist in Listing 1 als Pythoncode verdeutlicht.

doi: 10.2313/NET-2014-08-1_02

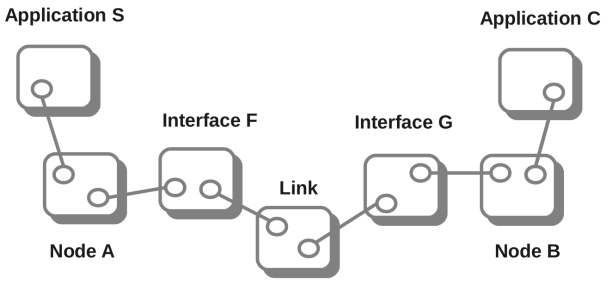

Abbildung 4: Topologischer Graph eines beispielhaften NEPI-Experiments [3]

```
from nepi.execution.ec import
    ExperimentController

ec = ExperimentController(exp_id="my_experiment
    ")
...
app_S = ec.register_resource("LinuxApplication"
    )
ec.set(app_s, "command", my_command)
node_A = ec.register_resource("LinuxNode")
ec.set(node_A, "hostname", my_hostname)
ec.set(node_A, "username", my_username)
...
ec.register_connection(app_S, node_A)
```

Listing 1: Code als Teil eines NEPI-Skripts, der einen Teilausschnitt von Abbildung 4 darstellt

4.3 Abhängigkeiten

Um den ereignisbedingten Ablauf des Experiments aufzuzeigen ist neben dem "Boxes and Connectors"-Diagramm noch ein zusätzlicher Abhängigkeitsgraph nötig. Dieser Graph gibt an, welche Bedingungen erfüllt sein müssen, bevor bestimmte Aktionen ausgeführt werden sollen. Die möglichen Aktionen sind das Deployment für eine Ressource und das Starten und Stoppen derselben. Dabei spielt der in Kapitel 4.1 genannte Status der Ressourcen eine entscheidende Rolle.
In Abbildung 5 ist ein solcher Abhängigkeitsgraph dargestellt. Applikation C soll gestartet werden, nachdem Applikation S gestartet wurde.
In Listing 2 ist die gleiche Abhängigkeit gezeigt, wie sie im Pythonskript aussieht.
[3]

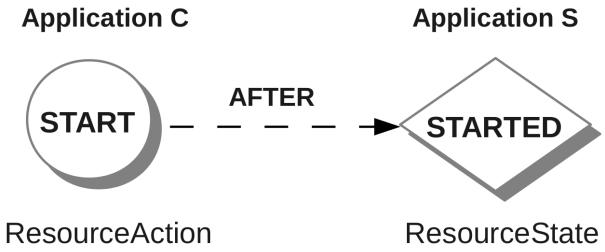

Abbildung 5: Abhängigkeitsgraph zweier Ressourcen [3]

```
from nepi.execution.resource import
    ResourceAction, ResourceState

ec.register_connection(app_C, ResourceAction.
    START, app_S, ResourceState.STARTED)
```

Listing 2: Pythoncode für NEPI, um Abhängigkeiten zu erzeugen

5. DURCHFÜHRUNG VON EXPERIMENTEN IN NEPI

Allein die Beschreibung eines Experiments ist noch nicht ausreichend für eine Durchführung. Für jede Testumgebung muss genau definiert sein, wie Software auf die beteiligten Ressourcen kommt, wie deren Steuerung erfolgt und wie die Testergebnisse gesammelt werden.
Ein Sensorknoten hat aber im Gegensatz zu einem Linux Host keinen SSH-Server, der einen einfachen Login zur Steuerung der Ressource ermöglicht. Deshalb muss für jede Testumgebung ein passender Resource Manager (RM) vorhanden sein der die Steuerung einer spezifischen Ressource übernimmt. Eine Besonderheit von NEPI ist, dass der Benutzer selbst einen Resource Manager für eine neue Testumgebung schreiben kann, der nur die gemeinsame Schnittstelle teilen muss.
Der User interagiert nur mit dem Experiment Controller (EC), der die Schnittstellenfunktionen der Resource Manager nutzt. [4]
In Abbildung 6 ist diese Abstraktion verdeutlicht. Links oben ist der User, der nur den Experiment Controller anspricht, welcher wiederum die Resource Manager über die gemeinsame Schnittstelle (in der Abbildung als "RM API" gekennzeichnet) steuert. Hier ist die Modularität von NEPI deutlich zu sehen. Jeder Resource Manager steuert seine eigenen Ressourcen und ein neuer RM kann problemlos hinzugefügt werden, was in Abbildung 6 als "API x" gekennzeichnet ist.

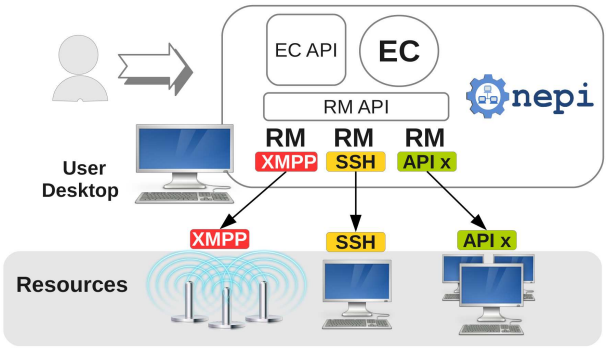

Abbildung 6: Resource Manager kennen die Steuerungsmöglichkeiten ihrer Testumgebungen [3]

Die Ausführung eines Experiments in NEPI kann in drei Phasen aufgeteilt werden, die in Abbildung 7 dargestellt sind. Diese Schritte sollen im Folgenden genauer erläutert werden.

doi: 10.2313/NET-2014-08-1_02

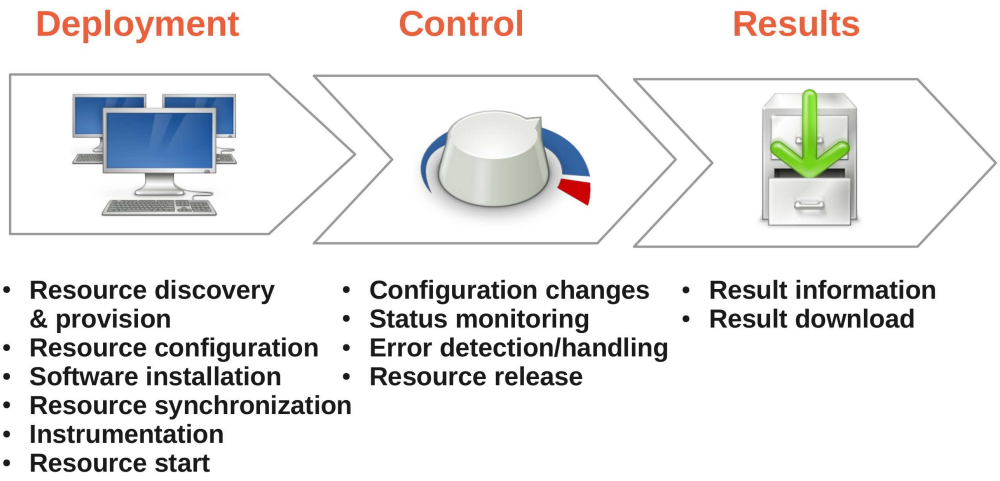

Abbildung 7: Verschiedene Phasen des Lebenszyklus eines Experiments [2]

5.1 Deployment

Das *Deployment* eines Experiments ist das Verteilen der Software auf die Ressourcen und deren Bereitstellung und Initialisierung. Das Deployment wird plattformabhängig von den Resource Managern gehandhabt, d.h. jeder Resource Manager muss wissen, mit welchen Protokollen und Authentifizierungen auf seine Ressourcen zugegriffen werden kann. Der Experiment Controller ruft nur eine `deploy`-Funktion auf, die alle Resource Manager gemeinsam haben.
Um eine Ressource für das Experiment nutzbar machen zu können müssen mehrere Schritte durchgeführt werden. Zunächst muss NEPI sicherstellen, dass die Ressource überhaupt vorhanden und erreichbar ist. Danach kann die Ressource konfiguriert werden, wozu auch die Installation fehlender Software gehört. Sind diese Schritte abgeschlossen versucht NEPI die Ressourcen zu synchronisieren und zu instrumentieren. Anschließend kann die Ressource gestartet werden. [Abbildung 7, links]

5.2 Steuerung

Der Benutzer kann das Experiment nicht nur vor dessen Beginn konfigurieren, sondern auch während der Ausführung noch Änderungen vornehmen. Dazu gehört auch das Hinzufügen von Ressourcen zur Laufzeit, wie bereits in Kapitel 4.1 erwähnt.
Der Benutzer hat die Möglichkeit im laufenden Experiment schon Traces von den Ressourcen herunterzuladen. Sie sind dort gespeichert wo sie erzeugt werden und der zentrale Experiment Controller muss die Traces explizit durch die Resource Manager von den potentiell entfernten Ressourcen anfordern. Diese Traces können zur Statusabfrage und Fehlererkennung genutzt werden. Dazu muss die Ressource im gestarteten Zustand sein. [3]
Ein Experiment wird mit dem Freigeben aller Ressourcen beendet. [Abbildung 7, Mitte]

5.3 Ergebnissammlung

Auch am Ende des Experiments können Traces gesammelt werden. Das ist vor allem für nicht-interaktive Experimente wichtig, die dem Nutzer keine Möglichkeit der Kontrolle während der Laufzeit bieten. Die Traces können von verschiedenen Testumgebungen heruntergeladen werden, was der jeweilige Resource Manager übernimmt.
Je nach benutzten Ressourcentypen gibt es unterschiedliche Arten von Traces. Eine Linux-Applikation hat die Standardausgabe und die Fehlerausgabe, während eine Netzwerkkarte Paketdumps mitschneiden kann. Es ist die Aufgabe des Benutzers aus den vorhandenen Traces die wichtigen auszuwählen, da NEPI über deren Relevanz nicht entscheiden kann. [Abbildung 7, rechts]

6. UNTERSTÜTZTE TESTBEDS

Ein zentraler Aspekt von NEPI ist die Fähigkeit verschiedene Testbeds zu unterstützen. Die Experimente sind nicht beschränkt auf eine einzige Testumgebung, sondern können mit kleinen Anpassungen des Skripts auch auf anderen Testumgebungen ausgeführt werden können.
Im Folgenden werden einige Testbeds vorgestellt, die aktuell, oder zukünftig von NEPI unterstützt werden.

6.1 PlanetLab

PlanetLab ist ein öffentlich zugängliches globales Netzwerk aus tausenden von Hosts, das für Netzwerkexperimente genutzt werden kann. Das Netzwerk ist an das Internet angebunden, und kann somit problemlos von Forschern und Entwicklern benutzt werden, um ihre Experimente unter realen Bedingungen zu testen.
Der Zugriff auf einzelne Nodes erfolgt per SSH mit Schlüsselauthentifizierung. Der Benutzer kann, im Gegensatz zu einem Simulationstestbed, keine beliebigen IP-Adressen an die PlanetLab Hosts vergeben, sondern nur die fest zugeteilten Adressen verwenden. Die Geräte sind nicht für ein einziges Experiment reserviert, es können gleichzeitig mehrere Experimente von unterschiedlichen Entwicklern laufen.
Im "Boxes and Connectors"-Diagramm wird ein PlanetLab Host als `Node`-Box verbunden mit einer `Internet`-Box dargestellt. In Abbildung 9 ist dies dreimal zu sehen. [2, 5]

6.2 OMF

Das c*O*ntrol and *M*anagement *F*ramework kurz OMF ist ein Framework für Netzwerktestbeds. Dabei geht es in erster Linie um WiFi-Testbeds. OMF erlaubt im Gegensatz zu PlanetLab nur einen Benutzer pro Host gleichzeitig, ermöglicht es den Entwicklern dadurch aber, den Host beliebig zu

doi: 10.2313/NET-2014-08-1_02

konfigurieren und anzupassen.
Die einzelnen Hosts werden über das XMPP-Protokoll konfiguriert, was für NEPIs Resource Manager eine Rolle spielt. [6]

6.3 ns-3 (zukünftig)

Der Netzwerksimulator ns-3 erlaubt präzise Kontrolle über viele Aspekte der simulierten Hosts und Geräte. Dabei kann die Simulation auf hardwarenahem Level erfolgen. Der Simulator wurde hauptsächlich zu Forschungs- und Bildungszwecken entwickelt, und ist deshalb öffentlich verfügbar. [7]
ns-3 wurde von NEPI v2.0 bereits vollständig unterstützt, in Version 3.0 ist die Einbindung aber noch nicht abgeschlossen. [8]
Die Anbindung an andere teilnehmende Ressourcen des Experiments erfolgt über eine `FileDescriptorNetDevice`-Ressource. Dieses Gerät kann Ethernetframes von einem Linux-Filedescriptor lesen und in Frames übersetzen, die im restlichen Testbed verarbeitet werden können. [1]

6.4 Nemu (zukünftig)

Nemu ist ein Netzwerkemulator, der vom gleichen Team entwickelt wurde, wie NEPI selbst. Der Emulator besteht aus zwei Hauptbestandteilen, netns und netem.
netns wurde bereits in NEPI v2.0 unterstützt, der Support der Kombination von netns und netem durch NEPI v3.0 ist in Arbeit. [8]
netns verwendet virtuelle Maschinen und Interfaces die miteinander verbunden sind. Außerdem wird der Netzwerkstack des Hostbetriebssystems von den virtuellen Maschinen geteilt.
Dadurch hat ein NEPI-Entwickler sehr viele Kontrollmöglichkeiten für das Experiment.[5, 1]

7. KOMBINIERTE VERWENDUNG

Eine Besonderheit von NEPI ist die Möglichkeit, hybride Experimente zu realisieren. Es können mehrere verschiedene Testbeds in einem Netzwerkexperiment kombiniert werden, wodurch beliebige Interaktionen ermöglicht werden.
In den Abbildungen 8 und 9 ist ein Experimentdesign beschrieben, das diese Eigenschaft nutzt. In diesem Experiment werden zwei verschiedene Testbeds benutzt. Rot hinterlegt (Oval, mittig) ist die PlanetLab-Testumgebung, die das Internet simuliert und drei WiFi-Netzwerke, je gelb(links oben), blau (rechts oben) und grün (unten) hinterlegt, die miteinander über das PlanetLab-Netz verbunden sind.
Die drei WiFi-Netzwerke laufen in je einem ns-3 Simulator. Die ns-3-Netzwerke enthalten Hosts, die realistischen Traffic durch VLC-, BitTorrent- oder VoIP-Anwendungen erzeugen.

8. ZUSAMMENFASSUNG

NEPI dient als Werkzeug, um Netzwerkexperimente zu vereinfachen. Die Probleme, die sich durch die Verwendung verschiedener Testbeds und die Komplexität der Ausführungsschritte eines Experiments ergeben, werden durch NEPIs zentralem Management deutlich verringert. Die Wiederholbarkeit der Experimente und die große Anzahl an Einsatzszenarien spricht für NEPI.
Probleme, die das Arbeiten mit NEPI erschweren sind einerseits das Fehlen von NEF, dem grafischen Frontend zum Erstellen der ”Boxes and Connectors”-Diagramme sowie die

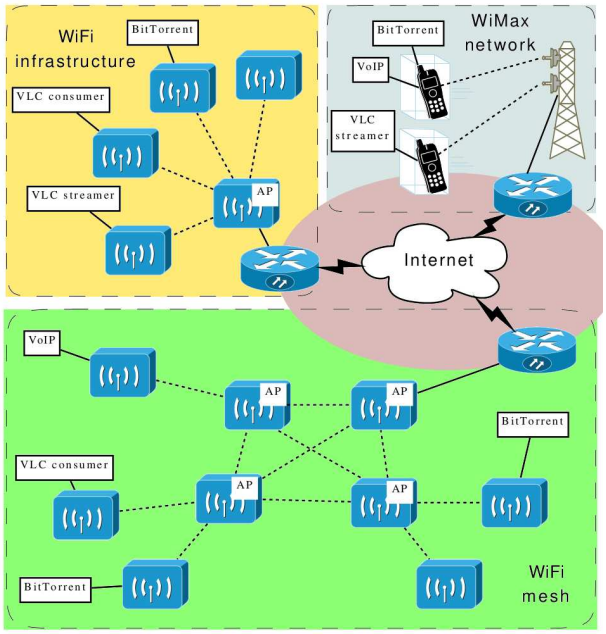

Abbildung 8: Darstellung eines hybriden Experiments [7]

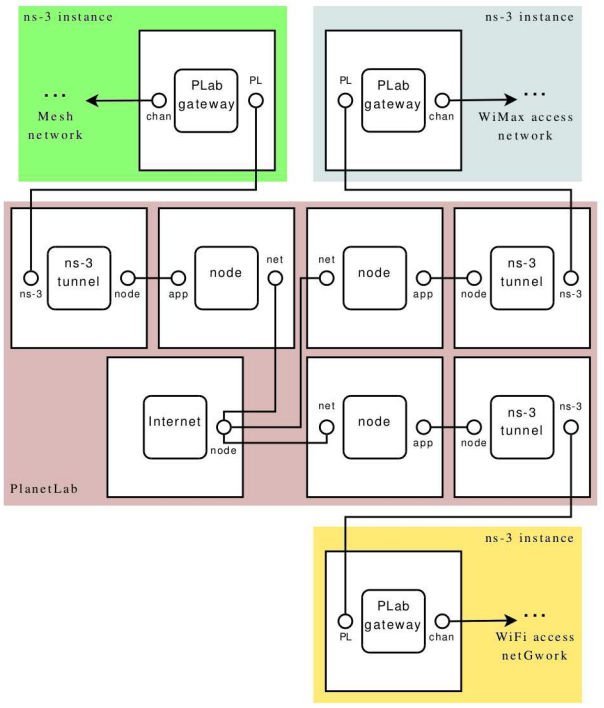

Abbildung 9: Ausschnitt des ”Boxes and Connectors”-Diagramms des Experiments aus Abbildung 8 [7]

doi: 10.2313/NET-2014-08-1_02

nicht abgeschlossene Entwicklung von NEPI.
Das zeigt sich darin, dass Publikationen zu Version 3.0 von NEPI nicht vorhanden sind, und die Internetpräsenz des Entwicklungsteams während der Erstellung dieses Papers ständig unter Bearbeitung war. [9] Die einzige Quelle, die die aktuellste Version behandelt ist das ausführliche, aber nicht abgeschlossene User Manual. [3]
Es bleibt zu hoffen, dass diese kleineren Schwachpunkte mit dem neuen Release, geplant für April/Mai 2014, beseitigt werden.

9. LITERATUR

[1] A. Quereilhac, et al.: *NEPI: An integration framework for network experimentation*, In Software, Telecommunications and Computer Networks (SoftCOM), 19th International Conference on. IEEE, 2011

[2] *NEPI hands-on* `http://nepi.inria.fr/pub/Nepi/WebHome/hands_on_nepi.pdf`, [Online, zugegriffen 18. März 2014]

[3] *NEPI v3.0 User Manual* `http://nepi.inria.fr/code/nepi/raw-file/68fabc2fc452/doc/user_manual/user_manual.pdf`, [Online, zugegriffen 17. März 2014]

[4] *Validating ICN implementations* `http://nepi.inria.fr/pub/Nepi/WebHome/NEPI_validating_ICN_implementations.pdf`, [Online, zugegriffer 17. März 2014]

[5] J. P. Rueda: *Testing NEPI usability and features*, Juni 2012

[6] M. Lacage: *Outils d'Expérimentation pour la Recherche en Réseaux*, PhD dissertation, November 2010

[7] M. Lacage, et al.: *NEPI: Using Independent Simulators, Emulators and Testbeds for Easy Experimentation*, In ROADS'09 workshop, Oktober 2009

[8] *Supported platforms* `http://nepi.pl.sophia.inria.fr/Technologies/WebHome`, [Online, zugegriffen 18. März 2014]

[9] *NEPI: Network Experiment Programming Interface* `http://nepi.inria.fr/Main/WebHome`, [Online, zugegriffen 23. März 2014]

doi: 10.2313/NET-2014-08-1_02