INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Ein Steuerungsprotokoll zur verbindungserhaltenden Migration virtueller Maschinen im WAN

Markus Jürgens

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Ein Steuerungsprotokoll zur verbindungserhaltenden Migration virtueller Maschinen im WAN

Markus Jürgens

Aufgabensteller:	PD Dr. Vitalian Danciu
Betreuer:	Tobias Guggemos PD Dr. Vitalian Danciu
Abgabetermin:	11. Juni 2018

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 11. Juni 2018

. . . . . .
(Unterschrift des Kandidaten)

Kurzfassung

Diese Arbeit behandelt das Thema der verbindungserhaltenden Migration virtueller Maschinen im WAN. Es existieren bereits Implementierungen für eine Live-Migration innerhalb einer LAN Umgebung. Ziel dieser Arbeit ist es, ein Protokoll zu entwerfen, das virtuelle Maschinen ohne Downtime verbindungserhaltend migriert und dabei nicht an die Grenzen des LANs gebunden ist. Um das zu erreichen, müssen die TCP-Verbindungen der VM zu Clients erhalten werden. Da sich die IP-Adresse der VM bei der Migration zwischen fremden Netzen ändert, ist dies mit reinem TCP nicht möglich. Um die Verbindungen trotzdem zu erhalten, wird MPTCP (Multipath TCP) verwendet. Mit MPTCP ist es möglich einer bestehenden TCP-Verbindung einen oder mehrere zusätzliche Pfade hinzuzufügen. So kann die VM mit der selben MPTCP-Verbindung über zwei IP-Adressen erreicht werden. Das in der Arbeit entwickelte Protokoll steuert die zu diesem Zwecke notwendige Kommunikation, sowie die erforderlichen Operationen auf dem Quellhost, dem Zielhost und der VM. Zusätzlich wurde dieses Protokoll im Zuge der Arbeit implementiert, um ein „Proof of Concept“ zu erstellen, das zeigt, dass eine verbindungserhaltende Migration virtueller Maschinen im WAN realisierbar ist. Das Ergebnis der Arbeit ermöglicht es beispielsweise Cloudanbietern, ihre physischen Hosts flexibler zu verwalten, indem VMs über Netzgrenzen hinweg migriert werden können, ohne dass die Kunden davon etwas mitbekommen.

Abstract

This thesis is about live migration of virtual machines on the WAN. There are existing implementations of live migration of virtual machines inside a LAN environment. This work develops a protocol that is capable of migrating a virtual machine between two hosts that are not located in the same broadcast domain. This migration will be without any downtime and it will preserve existing TCP-Connections of the VM to any clients. Migrating a VM between different networks means the VM will have a different IP address after the migration. Classic TCP will not be able to preserve a connection, if one party changes its IP address. Therefore this work will make use of MPTCP (Multipath TCP) to tackle this issue. MPTCP is capable of adding additional paths to existing TCP connections. This results in the VM being reachable via one MPTCP connection but more than one IP address. The developed protocol controls all the messages being exchanged between the source host, the destination host and the VM itself, as well as the operations that need to be applied to one of those three machines. Additional to developing the protocol a prototype was implemented as proof of concept, to demonstrate the ability to migrate VMs on the WAN without losing established TCP-Connections. The result of this work enables for example cloud service providers to maintain their physical hosts more flexible by migrating VMs not only to other machines on the LAN but on the WAN. During that whole migration process the customer does not even recognize the migration of the VM.

Inhaltsverzeichnis

1 Einleitung

Bereits seit den 1990er Jahren tauchte die Idee des „Cloud-Computings“ auf. 1993 sagte Eric Schmitd, langjähriger CEO von Google, voraus, dass mit ausreichender Bandbreite „der Computer ausgehöhlt [werde] und sich über das Netzwerk [verteile]“. [Rod08]. Trotzdem schaffte das Cloud-Computing den Durchbruch erst gegen Mitte bis Ende der 2000er. Mit ein Grund dafür ist der technologische Fortschritt im Bereich der virtualisierten Systeme, der das Anbieten von Cloud-Diensten auf administrativer Ebene deutlich erleichtert. Virtualisierung ermöglicht es, mehrere Systeme gleichzeitig auf einer Hardware zu betreiben. Dadurch erreicht man eine sinnvolle Kapselung der Daten und Zuordnung von Hardware-Ressourcen. Letztere kann auch im laufenden Betrieb an sich ändernde Anforderungen angepasst werden. Zusätzlich kann im Falle eines fatalen Fehlers in der virtuellen Maschine die Hardware des Anbieters nicht beschädigt werden. Diese Entwicklungen führten letztendlich dazu, dass das Anbieten von Cloud-Diensten ein weit verbreitetes Geschäftsmodell wurde und aus dem heutigen IT-Alltag nicht mehr wegzudenken ist. Allerdings steigen mit der Anzahl der Nutzer von Cloud-Diensten ebenso die Anforderungen an die Administration dieser Dienste. Wenn ein Kunde beispielsweise mehr Rechenleistung benötigt, so muss die virtuelle Maschine möglicherweise auf eine andere Hardware umgezogen werden, um die Anforderungen des Kunden zu erfüllen. Ein Umzug einer VM zwischen zwei verschiedenen Hosts ist auch sinnvoll, um eine bessere Lastverteilung zu erreichen. So kann es nützlich sein, VMs von einer physischen Maschine weg zu migrieren, um diese komplett abschalten zu können, oder um beispielsweise die Auslastung der Netzwerkkarte zu reduzieren.

Das Szenario für die Migration einer VM von einem Quellhost auf einen Zielhost, wird nun genauer betrachtet. Befinden sich diese beiden Hosts in dem gleichen Netz, so existieren bereits zahlreiche Lösungen, die VM ohne eine Verbindungsunterbrechung zu migrieren. Bei diesen Lösungen reicht es aus, die ARP-Tabellen auf der Sicherungsschicht des OSI-Modells zu aktualisieren. Die migrierte VM bleibt über ihre ursprüngliche IP-Adresse erreichbar. Das Routing, das auf der darüber liegenden Vermittlungsschicht stattfindet, muss hierbei nicht angepasst werden, bestehende TCP/IP-Verbindungen werden dabei nicht getrennt. Dieser Ansatz funktioniert allerdings nur, wenn sich sowohl Quellhost, als auch Zielhost in einem LAN befinden. Sobald diese Voraussetzung nicht mehr gegeben ist, muss das Routing angepasst werden. Das bedeutet, dass die zur VM bestehenden TCP/IP-Verbindungen abgebrochen werden.

Wenn Quellhost und Zielhost in unterschiedlichen Broadcast-Domänen beheimatet sind, so ist das Migrieren einer VM immer noch möglich. Allerdings ist die VM nach der Migration nicht mehr über ihre ursprüngliche IP-Adresse erreichbar. Demnach muss das oben angesprochene Routing verändert werden. Das wiederum führt dazu, dass mit aktuellen Methoden der Migration alle bestehenden Verbindungen von Clients zur VM getrennt und neu aufgebaut werden müssen.

1.1 Mehrwert einer Live-Migration im WAN

Die verbindungserhaltende Migration oder auch Live-Migration einer VM zeichnet sich dadurch aus, dass die VM ohne eine Trennung der bestehenden Verbindungen zu Clients migriert wird. Das hat den Vorteil, dass die Clients, die den Dienst der VM in Anspruch nehmen, keinen Verbindungsabbruch wahrnehmen und somit auch nicht benachrichtigt werden müssen, ob und wann eine VM migriert wird. Dadurch wird der administrative Aufwand eine VM zu migrieren erheblich verringert. Darüber hinaus wird ein zeitintensiver Up- oder Download-Vorgang bei der verbindungsorientierten Migration nicht unterbrochen und muss demnach auch nicht neu gestartet werden. Hierbei wird nicht nur Zeit eingespart, sondern auch die Kosteneffizienz aufgrund niedrigerer Netzauslastung gesteigert.

Nachdem die Vorteile durch den Erhalt der Verbindungen aufgezeigt wurde, werden nun die Vorteile, resultierend aus der Erweiterung, auf das WAN erörtert. Wie der Name „Local Area Network“ (LAN) schon aussagt, ist ein LAN in der lokalen Ausdehnung sehr begrenzt. Meist befinden sich alle Teilnehmer in einem Gebäude oder Rechenzentrum. Somit ist eine Migration von VMs im LAN eben an diese lokalen Grenzen gebunden. Mit der Migration im WAN ist es möglich, VMs über große geographische Distanzen zu migrieren. Dies eröffnet beispielsweise die Möglichkeit, ein komplettes Rechenzentrum vom Netz zu nehmen, bei gleichzeitiger Aufrechterhaltung aller zur Verfügung gestellten Dienste. Das macht dann Sinn, wenn am Standort eines Rechenzentrums ein starkes Unwetter oder eine andere Naturkatastrophe erwartet wird. Aber auch weniger katastrophale Ereignisse profitieren von der geographischen Unabhängigkeit einer Migration. Für ein global agierendes Unternehmen besteht so die Möglichkeit, auf die geographische Dichte der Anfragen an einen Dienst zu reagieren. Eine VM mit einem Dienst könnte so zum Beispiel in einem Rechenzentrum im asiatischen Raum starten und mit zunehmender Uhrzeit auf ein Rechenzentrum in Europa und anschließend nach Amerika migriert werden. So könnte sichergestellt werden, dass die VM zu jedem Zeitpunkt die meisten Anfragen von Clients in näherer Umgebung bekommt. Dies führt zu geringerer Auslastung transkontinentaler Leitungen und damit auch zu geringeren Kosten für den Dienstanbieter, der für die Leitungen relativ zur Auslastung durch seinen Dienst bezahlen muss.

1.2 Fragestellung und Ergebnisse

Ziel dieser Arbeit ist es, ein Protokoll zu entwerfen, das die Anforderungen an eine verbindungserhaltende Migration von VMs im WAN ermöglicht. Ein weiteres Ziel ist die Erstellung eines Prototypen, der dieses Protokoll implementiert und als ein „Proof Of Concept“ dient. Dieser soll demonstrieren, dass das Steuerungsprotokoll realisierbar ist und eine Migration von VMs über Netzgrenzen hinweg möglich ist, ohne dabei die TCP/IP-Verbindungen zu Clients zu trennen.

Das Ergebnis der Arbeit ist die Beschreibung des entwickelten Steuerungsprotokolls auf Basis von MPTCP. Ergänzend wurde ein Prototyp entwickelt, der in der Lage ist, die an das Protokoll gerichteten Anforderungen zu erfüllen. Dieser Prototyp zeigt in einer Laborumgebung, dass es möglich ist, VMs verbindungserhaltend im WAN zu migrieren.

1.3 Vorgehensweise und Struktur der Arbeit

Zu Beginn der Arbeit werden Anforderungen an die Funktionalität des Protokolls gesammelt. Anschließend wird die Fachliteratur nach Ansätzen zur verbindungserhaltenden Migration im WAN durchsucht, um bereits existierende Lösungen eventuell wiederverwenden zu können. Dabei wurde auch ein besseres Verständnis für die Problemstellung erlangt. Anhand der gewonnenen Erkenntnisse ist es dann möglich, einen Versuchsaufbau zu erstellen. Dieser besteht aus Quellhost, Zielhost einer VM auf dem Quellhost und einem Client. Die VM muss einen MPTCP-Kernel mit gepatchter Socket-API zur Steuerung der MPTCP-Pfade besitzen. Der Client muss ebenfalls über einen MPTCP-Kernel besitzen. Mit diesem Versuchsaufbau und einer Server-Client-Anwendung kann dann die Funktionalität von MPTCP sichergestellt und nachvollzogen werden. Daraufhin wird versucht, eine verbindungserhaltende Migration im WAN durchzuführen. Zu diesem Zweck wird eine bereits existierende Lösung zur Migration von virtuellen Maschinen verwendet. Diese Art der Migration stellt allerdings keinen Verbindunserhalt im WAN sicher. Mit MPTCP und der expliziten Steuerung der Pfade, kann diese Anforderung trotzdem realisiert werden. Zu diesem Zeitpunkt ist es erstmalig möglich, eine VM verbindungserhaltend über Netzgrenzen hinweg zu migrieren. Nun muss der Informationsaustausch zwischen Quellhost, Zielhost und VM, der notwendig ist, um die Verbindungen zu erhalten, festgehalten werden und von dem Rest des Migrationsvorgangs abstrahiert werden. Dieser Informationsaustausch wird dann zu Protokollfunktionen generalisiert. Außerdem werden die Rollen der beteiligten Maschinen festgelegt. Der nächste Schritt ist dann die Spezifizierung des Protokolls. Mit dieser kann dann eine prototypische Implementierung erstellt werden, die den Protokollanforderungen genügt und eine VM verbindungserhaltend über Netzgrenzen hinweg migriert. Der letzte Schritt ist die Evaluierung des erstellten Prototyps in der Laborumgebung.

Kapitel 2 schildert einen konkreten Anwendungsfall und leitet ausgehend von diesem Anforderungen an das Protokoll ab.

Kapitel 3 zeigt den aktuellen Stand der Technik auf. Hierbei wird die verbindungserhaltende Migration im WAN diskutiert, sowie ein Blick auf eine bestehende Technologie zur Live-Migration im WAN geworfen. Anschließend werden Multipath-TCP und die sich bietenden Möglichkeiten beschrieben.

Kapitel 4 erklärt das im Zuge dieser Arbeit erstellte Steuerungsprotokoll für eine verbindungserhaltende Migration von VMs im WAN. Dabei wird der Nachrichtenaustausch im zeitlichen Kontext beschrieben, gefolgt von der Erläuterung der Nachrichteninhalte. Dann wird das Protokoll mittels Zustandsdiagrammen visualisiert. Am Ende des Kapitels findet sich der Umgang mit Fehlern, die potenziell im Protokollablauf auftreten.

Kapitel 5 befasst sich mit dem Erstellen des Prototypen. An dieser Stelle wird zuerst die Laborumgebung beschrieben und es werden verwendete Protokolle und Programme genannt. Anschließend wird die Programmstruktur auf Quellhost, Zielhost und auf der VM aufgezeigt.

Kapitel 6 erklärt zunächst das auf VM und Client laufende Programm „TCP-Ping-Pong“, das in dieser Arbeit die ununterbrochene Erreichbarkeit der zu migrierenden VM im Migrationsprozess zeigt. Anschließend werden die Beobachtungen und Ergebnisse des Migrationsprozesses in der Laborumgebung festgehalten und ausgewertet. An dieser Stelle findet ein Abgleich mit den in Kapitel 2 gestellten Anforderungen statt.

Kapitel 7 fasst die Ergebnisse der Arbeit in einem Fazit zusammen. Außerdem gibt es einen Ausblick auf mögliche weiterführende Arbeiten zu dem Thema.

2 Anforderungsanalyse

In diesem Kapitel wird zuerst ein beispielhaftes Szenario gezeigt, das von einem Steuerungsprotokoll zur verbindungserhaltenden Migration virtueller Maschinen profitiert. Anschließend werden anhand dieses Szenarios Anforderungen definiert, die von dem Steuerungsprotokoll erfüllt werden müssen.

2.1 Anwendungsszenario

Um den Mehrwert eines Steuerungsprotokolls zur verbindungserhaltenden Migration von virtuellen Maschinen im WAN zu veranschaulichen, wird ein konkretes Anwendungsszenario beschrieben.

Ein IT-Dienstleister bietet mehrere Dienste über das Internet an. Dazu betreibt er mehrere physische Maschinen, die zum einen Teil in Europa stehen, zum anderen Teil in den USA. Auf diesen Maschinen betreibt der Dienstleister VMs, in denen je ein Webserver läuft. Dieser Webserver wartet auf Verbindungsanfragen von Clients und bearbeitet diese. Jeder dieser Webserver kann mehrere Verbindungsanfragen parallel abarbeiten. Es wird davon ausgegangen, dass die Webserver Anfragen aus der ganzen Welt bekommen.

Die geographische Verteilung der Clients ist abhängig von der lokalen Uhrzeit der Clients. So ist es vorstellbar, dass die Webserver während später Nacht- und früher Morgenstunden in Europa weniger Anfragen von Clients aus Europa bekommen. Während dieser Zeit ist es in den USA Tag und die Webserver erhalten viele Verbindungsanfragen von amerikanischen Clients. Daher entscheidet sich der Dienstleister aus Gründen der Kosteneffizienz möglichst viele VMs mit den Webservern in die USA zu migrieren. Dadurch minimiert der Dienstleister die Nutzung von Transkontinentalkabeln und spart Kosten ein. Dem Dienstleister ist es aber nicht nur wichtig Kosten zu minimieren, sondern auch die Zufriedenheit seiner Kunden zu maximieren.

Die Migration von Webservern von Europa in die USA stellt eine Migration im WAN dar, die VM besitzt nach der Migration eine neue IP-Adresse. Demnach ist es dem Webserver aktuell nicht möglich die bestehenden Verbindungen zu den Clients zu erhalten. Jeder Client muss eine neue Verbindung zu dem migrierten Webserver aufbauen, unbearbeitete Anfragen gehen dabei verloren und müssen erneut angefragt werden. Der Austausch großer Datenmengen, beispielsweise ein großer Download, der einige Zeit in Anspruch nimmt wird von der Migration abgebrochen und muss nach der Migration von Neuem gestartet werden.

Für den Dienstleister wäre es wünschenswert die Verbindungen zu den Clients zu erhalten, um die genannten Probleme zu umgehen und den Clients eine bessere User Experience zu bieten. Für die Clients wäre die Migration dann komplett transparent. Das heißt, sie würden nicht bemerken, dass der Webserver, mit dem sie verbunden sind, seinen Host und seine IP-Adresse geändert hat.

Mit fortschreitender Zeit wird es wieder Sinn machen, so viele VMs wie möglich auf europäischen Servern zu hosten. Dann können diese wieder zurück nach Europa migriert

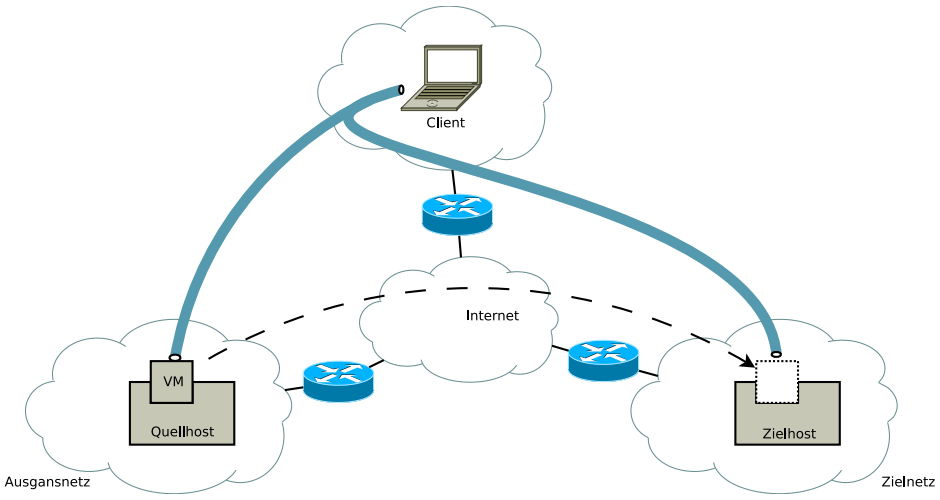

Abbildung 2.1: Anwendungsszenario

werden. Funktioniert dieser Prozess reibungslos und zuverlässig, so stellt das geschilderte Szenario eine elegante Möglichkeit dar, Kosten einzusparen, ohne dabei Kompromisse eingehen zu müssen.

Abbildung 2.1 zeigt das Szenario, bei dem eine VM von einem Quellhost im Ausgangsnetz auf einen Zielhost im Zielnetz migriert wird. Die Verbindung von Client zur VM vor der Migration stellt die TCP-Verbindung des Clients zur VM dar. Die Verbindung des Clients zur VM im Zielnetz stellt die **gleiche** TCP-Verbindung dar, lediglich führt diese nicht mehr in das Ausgangsnetz, sondern in das Zielnetz, wo sich die VM nach der Migration befindet. Wichtig ist es hierbei zu verstehen, dass der Client seine Datenpakete während des kompletten Migrationsvorgangs stets über die gleiche TCP-Verbindung schickt.

2.2 Anforderungen an das Protokoll

Aus dem oben beschriebenen Anwendungsszenario lassen sich konkrete Anforderungen ableiten, die bei der Entwicklung eines Protokolls zur verbindungserhaltenden Migration von VMs im WAN berücksichtigt werden müssen. Hierbei wird unterschieden zwischen funktionalen und nicht funktionalen Anforderungen.

2.2.1 funktionale Anforderungen

Erhalt der bestehenden TCP-Verbindungen: Das Protokoll muss in der Lage sein, die Migration einer VM in ein fremdes Netz zu steuern, ohne dabei die TCP-Verbindungen der Server-Anwendung innerhalb der VM abzubrechen. Hierbei ist zu berücksichtigen, dass die Server-Anwendung in der Lage sein muss, mehrere Clients zu bedienen. Somit muss die Server-Anwendung auch die bestehenden TCP-Verbindungen zu allen diesen Clients erhalten.

Sitzungsverwaltung: Um den Nutzen des Protokolls zu maximieren muss das Protokoll über eine sinnige Sitzungsverwaltung verfügen. Der Quellhost muss in der Lage sein, das Protokoll mehrmals zeitgleich ausführen zu können. Wenn mehrere VMs von

einem Host weg migriert werden sollen, muss der Administrator nicht warten bis eine Migration abgeschlossen ist, um eine neue anzustoßen, sondern er kann spezifizieren, welche VM zu welchem Zielhost migriert werden soll und diese Migrationen anstoßen. Ebenso soll ein Zielhost in der Lage sein, mehrere Protokollanfragen zur gleichen Zeit bearbeiten zu können. Wenn zwei Anfragen nahezu zeitgleich den Zielhost erreichen, so soll dieser beide Anfragen bearbeiten und die später ankommende Nachricht nicht bis zum Abschluss des Protokollablaufs der ersten Anfrage blockieren.
Die VM soll Anfragen einer zweiten oder weiteren Migration immer verwerfen, solange aktuell ein Migrationsvorgang läuft. Die VM kann nicht zu einem Zeitpunkt auf zwei Zielhost migriert werden. Wenn ein Migrationsvorgang abgeschlossen ist, soll die VM weitere Migrationsanfragen akzeptieren und bearbeiten.

Fehlererkennung und -rehabilitation: Das Protokoll muss das Auftreten von Fehlern erkennen und melden. Außerdem muss das Protokoll über eine Rehabilitation von aufgetretenen Fehlern verfügen. Das bedeutet, wenn ein Fehler aufgetreten ist, dann muss das Protokoll in einen Modus übergehen, der alle bis zu dem Zeitpunkt des Fehlers getätigten Änderungen wieder rückgängig macht und die beteiligten Maschinen in einem sauberen Zustand hinterlässt. Nur so kann sichergestellt werden, dass der Betrieb der Maschinen nach dem Auftreten eines Fehlers und keiner Migration weiterhin möglich ist.

Interface zur Server-Applikation: Das Protokoll muss Nachrichten mit der Server-Applikation austauschen. Nur die Server-Applikation ist im Stande, den Erhalt der bestehenden Verbindungen sicherzustellen. Die Server-Applikation muss demnach benachrichtigt werden, um sich darum zu kümmern und dann dem Protokoll mitteilen, dass die notwendigen Aktionen abgeschlossen wurden. Außerdem soll die Server-Applikation versuchen, vorauszusagen, wieviele der bestehenden Verbindungen voraussichtlich erhalten werden können.

2.2.2 Nicht-funktionale Anforderungen

Effizienz: Das Protokoll muss stets nach einem Minimalitätsprinzip arbeiten. Um einen schnellen und reibungslosen Ablauf des Protokolls zu ermöglichen sollte dies lediglich notwendige Operationen auf den Maschinen ausführen. Zusätzlich sollte der Nachrichtenaustausch auf ein notwendiges Minimum beschränkt werden. Die Größe der Nachrichten sollte ebenfalls gering gehalten werden. Der Netzverkehr sollte so gering wie möglich gehalten werden.

Robustheit: Das Protokoll muss Änderungen, die für den Ablauf notwendig sind, aber nach einem bestimmten Schritt nicht mehr notwendig sind, wieder rückgängig machen. Sowohl Quellhost als auch Zielhost müssen sich nach einer erfolgreichen Migration wieder in dem gleichen Zustand befinden, in dem sie vor der Migration waren. Natürlich mit Ausnahme, dass die VM nach einer geglückten Migration nicht mehr auf dem Quellhost, sondern auf dem Zielhost läuft. Auch die VM muss nach dem Protokollablauf wieder in einem Zustand sein, der sich von ihrem Anfangszustand nur in der Netzkonfiguration unterscheidet.

2.2.3 Zusammenfassung der Anforderungen

Tabelle 2.1 fasst die analysierten Anforderungen zusammen und versieht diese mit eindeutigen Identifikationsnummern. Diese erleichtern im weiteren Verlauf der Arbeit Bezug auf die Anforderungen zu nehmen.

ID	Anforderung
funktionale Anforderungen	
F1	Erhalt der bestehenden TCP-Verbindungen
F2	Sitzungsverwaltung
F3	Fehlererkennung und -rehabilitation
F4	Interface zur Server-Applikation
nicht-funktionale Anforderungen	
NF1	Effizienz
NF2	Robustheit

Tabelle 2.1: Anforderungen

3 Stand der Technik

Dieses Kapitel zeigt den aktuellen Stand der Technik. Der erste Abschnitt beschreibt die verbindungserhaltende Migration im LAN. Daraufhin wird die verbindungserhaltende Migration im WAN über IP-Tunnel mit Vor- und Nachteilen beschrieben. Der letzte Abschnitt in diesem Kapitel beschäftigt sich mit Multipath-TCP, einer Protokollerweiterung von TCP.

3.1 Verbindungserhaltende Migration im LAN

Bereits 2005 haben Clark et al. [CFH+05] ihre Arbeit zur Live-Migration virtueller Maschinen veröffentlicht. Da diese Technik, virtuelle Maschinen zu migrieren, auch in der vorliegenden Arbeit Anwendung findet, lohnt sich eine genauere Betrachtung. Clark unterteilt den Migrationsvorgang in 5 Abschnitte. Im ersten Abschnitt „Pre-Migration“ läuft die VM auf dem Quellhost. Im nächsten Abschnitt „Reservation“ wird sichergestellt, dass der Zielhost genügend Ressourcen besitzt, um die zu migrierende VM zu erhalten. Im nächsten Schritt „Iterative Pre-Copy“ werden Speicherseiten auf den Zielhost kopiert. Dieser Vorgang ist iterativ, findet also mehrmals statt, wobei nur in der ersten Iteration alle Speicherseiten kopiert werden. In den folgenden Iterationen werden lediglich die Speicherseiten auf den Zielhost kopiert, die sich im Vergleich zur vorigen Iteration geändert haben. Ein Algorithmus entscheidet über den besten Zeitpunkt, die „Iterative Pre-Copy“-Phase zu verlassen und in die „Stop-and-Copy“-Phase überzugehen. Sinnvollerweise findet dieser Übergang zu einem Zeitpunkt statt, zu dem die Speicherseiten, die sich nie oder nur selten ändern, schon auf den Zielhost kopiert wurden. Außerdem sollte in die „Stop-and-Copy“-Phase übergegangen werden, wenn sich die veränderten Speicherseiten zwischen den Iterationen schneller ändern, als eine Iteration Zeit benötigt. Im Abschnitt „Stop-and-Copy“ wird die VM auf dem Quellhost angehalten und die Speicherseiten, die sich im Vergleich zur letzten Iteration geändert haben werden ein letztes mal kopiert. Zusätzlich wird der Zustand der CPU kopiert und der Datenverkehr zum Zielhost umgeleitet. In dieser Phase können Datenpakete im Netz verloren gehen. Da diese Phase aber möglichst kurz gehalten ist, beeinträchtigen diese Verluste die Datenübertragung kaum. Im Abschnitt „Commitment“ bestätigt der Zielhost den erfolgreichen Erhalt der VM und der Quellhost verwirft die VM. Im letzten Abschnitt „Activation“ wird die VM auf dem Zielhost gestartet und der Umzug der IP-Adresse wird im Netz propagiert. Damit ist die Live-Migration der VM abgeschlossen.

Es ist wichtig zu verstehen, dass die Live-Migration nach Clark nur innerhalb einer LAN Umgebung stattfinden kann. Die VM besitzt vor und nach der Migration die gleiche IP-Adresse. Das ist zwingend notwendig, um die TCP/IP-Verbindungen der Clients zur VM aufrechtzuerhalten. So ist es ausreichend, nach der Migration den Umzug der IP-Adresse im Netz zu propagieren. Dies wird technisch durch das Aktualisieren der ARP-Tabellen im Netz realisiert, ein Vorgang, der im OSI-Modell auf der Sicherungsschicht stattfindet. Der Client, der eine TCP/IP-Verbindung zu der VM hat, muss an dieser Verbindung nichts ändern, da die IP-Adresse der VM unverändert bleibt. Demnach muss der Client auch keine Kenntnis

von der Migration haben, insbesondere ist weder ein Vorbereiten noch ein Nachbereiten der Migration auf Seiten des Clients notwendig.

3.2 Verbindungserhaltende Migration im WAN über IP-Tunnel

Es existieren bereits Lösungen, virtuelle Maschinen verbindungserhaltend im WAN zu migrieren. Diese arbeiten weitestgehend nach dem von Travostino et al. erarbeitetem Ansatz aus 2006 [TPDG+06]. Dieser wird im nächsten Unterabschnitt kurz beschrieben. Darauf folgt das Aufzeigen der Nachteile, den dieser Ansatz mit sich bringt.

3.2.1 Funktionsweise

Um eine Live-Migration im WAN durchführen zu können, muss das Routing nach der Migration angepasst werden. Die VM besitzt nach der Migration eine neue IP-Adresse. Um die Verbindungen von Clients, die vor der Migration mit der VM verbunden waren, zu erhalten, muss die VM allerdings unter der alten IP-Adresse erreichbar bleiben. Dies kann mit Hilfe von IP-Tunneln realisiert werden. Während des Migrationsvorgangs wird dann der Tunnelendpunkt der VM umkonfiguriert, so dass der Datenverkehr über den Zielhost geroutet wird und nicht mehr über den Quellhost. Gleichzeitig werden neue Anfragen an die VM direkt an die neue IP-Adresse umgeleitet. Dieser Schritt erfordert eine Änderung der DNS-Auflösung. Wenn alle Verbindungen, die über den Tunnel abgewickelt werden, beendet sind, kann der IP-Tunnel abgebaut werden.

3.2.2 Nachteile

Das Problem dieses Ansatzes liegt in der Tatsache, dass der Datenverkehr, der auf logischer Ebene durch den IP-Tunnel geleitet wird, physisch einen weiteren Weg zurücklegt. So läuft der Traffic zuerst bis zum Tunnelendpunkt im Ursprungsnetz. Diese Maschine mit dem Tunnelendpunkt kennt dann eine Route zu dem Tunnelendpunkt im Zielnetz. Von dort gelangt der Datenverkehr schließlich zur migrierten VM. Die Antwort der VM wird logisch wieder durch den IP-Tunnel geschickt. Das bedeutet, physisch nimmt der Datenverkehr die gleiche Route zurück zu dem Client. Dadurch bedingt gehen einige Aspekte der in Abschnitt 1.1 aufgeführten Mehrwerte verloren. Wenn beispielsweise ein Client aus Europa auf eine VM in der USA zugreift und diese VM nun in ein Rechenzentrum nach Europa migriert wird, so hat sich die Nutzung eines Transatlantikkabels verdoppelt. Zuerst wird der Datenstrom an das Rechenzentrum in der USA geschickt, um dort von einem Router wieder zurück nach Europa umgeleitet zu werden. Erst dann landet die Anfrage des Clients bei der gewünschten VM. Die Antwort der VM wird dann wiederum zuerst an das Rechenzentrum in den USA geschickt, um dann weiter nach Europa geroutet zu werden. Idealerweise würde die Anfrage des Clients gar nicht über ein Transatlantikkabel geführt werden, sondern die VM über einen direkteren Weg erreichen. Dieser Umstand ist auch nicht mit der nicht-funktionalen Anforderung NF1 (Effizienz) aus Kapitel 2 konform.

Ein weiterer in Abschnitt 1.1 aufgeführter Mehrwert, der bei diesem Ansatz nicht mehr realisierbar ist, ist die komplette Abschaltung eines Rechenzentrums. In diesem Falle kann der IP-Tunnel nicht mehr aufgebaut werden oder, wenn dieser bereits besteht, würde der Tunnel keine Daten mehr durchreichen, sobald die Maschine, die den Tunnelendpunkt im Ursprungsnetz betreibt, abgeschaltet wird.

3.3 MPTCP

Um eine verbindungserhaltende Migration im WAN durchführen zu können, ist es notwendig, dass eine aktive TCP/IP-Verbindung nicht abbricht, obwohl sich die IP-Adresse eines Kommunikationspartners ändert. Nachdem im vorhergehenden Abschnitt 3.2 erläutert wurde, warum der Ansatz über IP-Tunnel nicht jeder Art von Anforderung gerecht wird, zeigt dieser Abschnitt eine Technologie, mit der der angesprochene Mehrwert gewonnen wird. Das von Paasch und Bonaventure entwickelte Multipath-TCP ist in der Lage, genau diese Lücke zu füllen [PB14].

MPTCP ist ein Protokoll, das es ermöglicht, einer bestehenden TCP-Verbindung zusätzliche Subflows hinzuzufügen. Voraussetzung dafür ist, dass beide Kommunikationspartner in der Lage sind, MPTCP-Verbindungen zu handhaben. Ein zusätzlicher Subflow ermöglicht es, Daten über einen zweiten Pfad zu schicken. Dieser zweite Pfad ist charakterisiert durch die Sender-IP und den Sender-Port, sowie durch die Empfänger-IP und den Empfänger-Port. Um einen neuen Subflow zu öffnen, muss sich nur eine dieser vier Angaben zu einem bereits bestehenden Subflow unterscheiden. MPTCP wurde so entworfen, dass es einen mindestens gleich guten Datendurchsatz wie TCP hat. Darüber hinaus muss eine normale TCP-Verbindung etabliert werden können, wenn nicht beide Kommunikationspartner in der Lage sind via MPTCP zu kommunizieren. Außerdem soll es mit der von TCP-Verbindungen bekannten „Socket API“ steuerbar sein.

Der Verbindungsaufbau einer MPTCP-Verbindung erfolgt auf sehr ähnliche Art wie der einer regulären TCP-Verbindung (3-Way-Handshake). Allerdings schickt der erste Kommunikationspartner mit seiner `SYN` Nachricht die TCP-Option `MP_CAPABLE` mit. Ist der zweite Kommunikationspartner auch in der Lage über MPTCP zu kommunizieren, so fügt dieser ebenso die `MP_CAPABLE`-Option seiner `SYN+ACK` Nachricht an. Der erste Kommunikationspartner bestätigt den Aufbau einer MPTCP-Verbindung wiederum durch das Verwenden der `MP_CAPABLE`-Option in der `ACK` Nachricht. Ab diesem Zeitpunkt ist der erste Subflow einer MPTCP-Verbindung aufgebaut. Dieser Ablauf wird von Abbildung 3.1 veranschaulicht.

Nun kann einer bestehenden MPTCP-Verbindung ein zusätzlicher Subflow angefügt werden. Hierfür dient das `MP_JOIN`-Flag. Dieser Ablauf ähnelt dem initialen Verbindungsaufbau und ist in Abbildung 3.2 abgebildet.

Ab dem Moment, an dem es mehr als nur einen Subflow gibt, können beide Kommunikationspartner diese Subflows nutzen. Auch kann das selbe Paket über mehrere Subflows geschickt werden, um Datenverlust über einen Subflow entgegenzuwirken. Anhand der TCP Sequenznummern kann sichergestellt werden, dass die Datenpakete in der richtigen Reihenfolge an die darüberliegenden Schichten weitergegeben werden.

3.4 Zusammenfassung

Zusammenfassend lässt sich sagen, dass es bereits Lösungen zur Live-Migration gibt, die unterhalb der Vermittlungsschicht den Erhalt der bestehenden Verbindungen sicherstellt. Darüber hinaus existieren bereits Überlegungen, die den Erhalt der Verbindungen auch bei Migration im WAN sicherstellen. Diese Überlegungen genügen allerdings nicht den von dieser Arbeit an das Protokoll gestellten Anforderungen. Zusätzlich existiert eine Protokollerweiterung für TCP, die es erlaubt, Verbindungen zwischen mehreren Tupeln von IP-Adressen und Ports zu etablieren. Mithilfe dieser Protokollerweiterung ist es möglich, den Erhalt bestehender

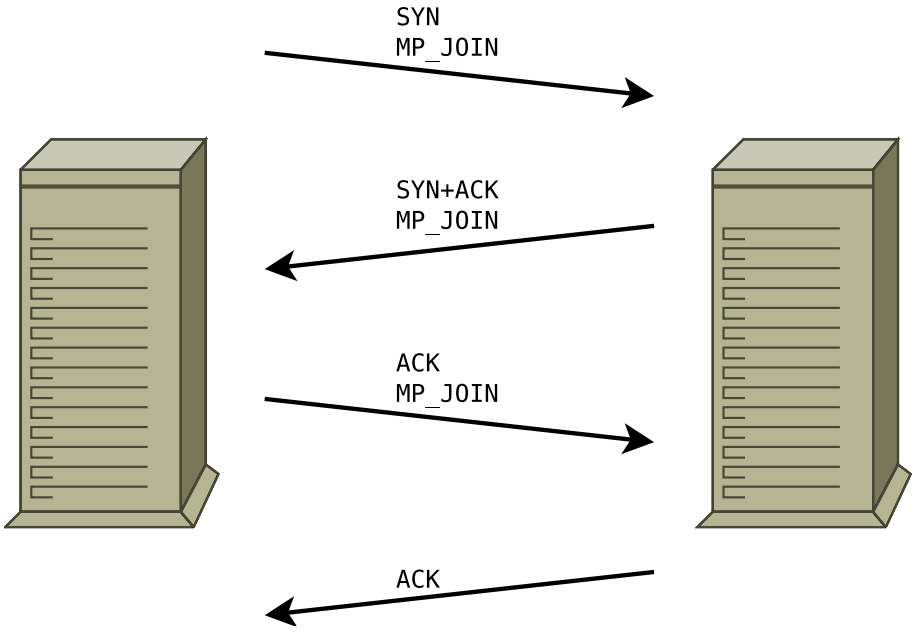

Abbildung 3.1: Verbindungsaufbau bei MPTCP, Abbildung nach [PB14, S. 3]

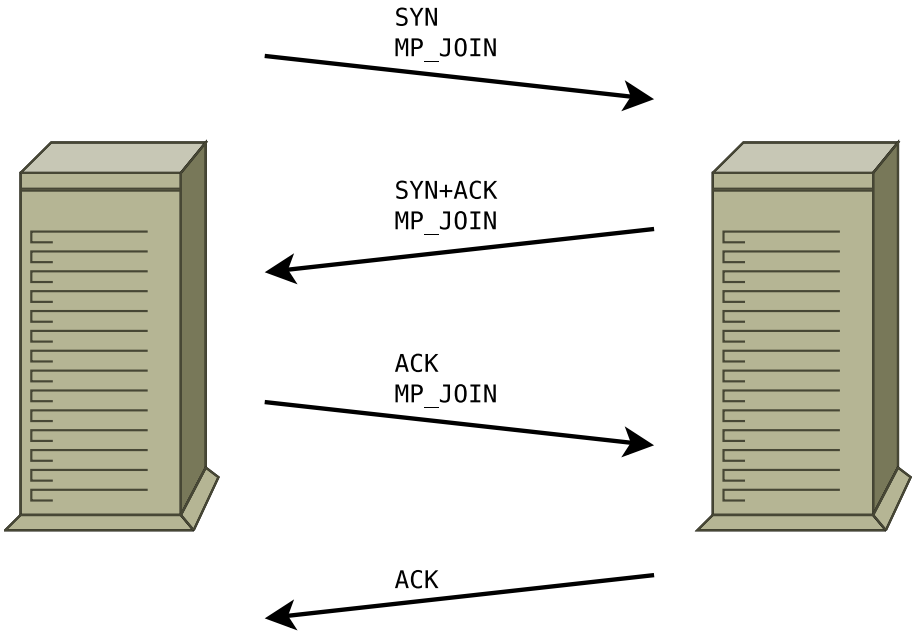

Abbildung 3.2: Hinzufügen eines Subflows, Abbildung nach [PB14, S. 4]

Verbindungen sicherzustellen, auch wenn eine VM über Netzgrenzen hinweg migriert wird.

4 Beschreibung des Steuerungsprotokolls

Im folgenden Kapitel wird das erarbeitete Protokoll beschrieben. Zuerst werden einige Designentscheidungen aufgeführt, gefolgt von der Beschreibung der Nachrichten im Kontext des zeitlichen Ablaufs. Dann wird der Inhalt einer Nachricht, eine sogenannten Protocol Data Unit erklärt. Diese beinhaltet die Informationien, die zwischen Quellhost, Zielhost und VM ausgetauscht werden. Im darauf folgenden Abschnitt werden die Zustände, die ein beteiligter Kommunikationspartner einnehmen kann, aufgezeigt. Am Ende des Kapitels beschäftigen wir uns mit dem Verhalten des Protokolls im Fehlerfall.

4.1 Designentscheidungen

Um einen geregelten und stabilen Ablauf des Protokolls zu gewährleisten und um eine möglichst komfortable Nutzung des Protokolls zu ermöglichen, wurden einige Designentscheidungen getroffen, die in folgender Auflistung geführt werden.

- **Fähigkeit des Quellhosts, mehrere VMs gleichzeitig zu migrieren:** Um einige der Anwendungsfälle, die in Abschnitt 1.1 genannt wurden, sinnvoll realisieren zu können, ist es notwendig, mehrere VMs gleichzeitig vom Quellhost hin zu einem oder mehreren Zielhosts migrieren zu können. Dafür benötigt man für jeden Migrationsvorgang eine eindeutige Migrations-ID. Damit können Nachrichten zwischen den Kommunikationspartnern eindeutig einem Migrationsprozess zugeordnet werden und somit kann man dann wiederum feststellen, ob Nachrichten in falscher Reihenfolge oder doppelt gesendet wurden.

- **Fähigkeit des Zielhosts, mehrere VMs gleichzeitig zu erhalten:** Damit der Zielhost mehrere VMs gleichzeitig empfangen kann, muss auch dieser die im obigen Punkt erwähnten Migrations-IDs nutzen. Zusätzlich muss der Zielhost auch in der Lage sein, Migrationsanfragen mit einer neuen Migrations-ID zu akzeptieren. Außerdem muss der Zielhost bei Nachrichten mit bereits bekannter Migrations-ID überprüfen, ob die Nachricht bezüglich der Reihenfolge die erwartete Nachricht ist.

- **Fähigkeit der VM, mehrfach migriert zu werden:** Die VM bearbeitet Nachrichten nur, wenn nicht aktuell ein Migrationsprozess abläuft, und wenn eine Nachricht mit der gleichen Migrations-ID, wie die der vorhergehenden Nachricht eintrifft. Die VM darf keine zwei Migrationsvorgänge gleichzeitig bearbeiten. Erst wenn ein Migrationsvorgang abgeschlossen ist, darf die VM eine Anfrage zu einem neuen Migrationsvorgang annehmen und bearbeiten. Jedoch soll die VM nach dem Beenden eines Migrationsvorgangs wieder in einen Zustand übergehen, der Migrationsanfragen bearbeitet.

- **Wiederherstellung des Ausgangszustandes bei fehlerhaftem Ablauf:** Wenn zu irgendeinem Zeitpunkt im Ablauf eines Migrationsvorgangs ein Fehler auftritt, der eine Migration unmöglich macht, so muss sichergestellt werden, dass sowohl Quellhost

als auch Zielhost und VM in einen sauberen Zustand gebracht werden. Alle Änderungen, die an einer der drei beteiligten Parteien vorgenommen wurden, müssen rückgängig gemacht werden. Zu diesem Zweck wird eine Fehler-ID verwendet, die den aufgetretenen Fehler nicht nur erkennt, sondern auch eine eindeutige Zuordnung erlaubt.

- **Übermittlung der MAC-Adresse zur IP-Adressgenerierung:** Um in zukünftigen Implementierungen des Protokolls die IP-Adressgenerierung über DHCP abwickeln zu können, wird die MAC-Adresse bei der Anfrage an den Quellhost nach einer freien IP-Adresse mitgeschickt.
- **Quote geöffneter Subflows:** Um eine Quote von Clients, deren Verbindung bei einer Migration erhalten bleibt, bestimmen zu können, verfügt das Protokoll über die Gesamtanzahl verbundener Clients und die Zahl an Clients, zu denen ein zweiter Subflow hinzugefügt werden konnte.

4.2 Zeitlicher Ablauf

Um die Funktionsweise des Protokolls eingängiger erläutern zu können, wird in diesem Abschnitt ein Blick auf die zeitliche Abfolge der Aktionen des Protokolls geworfen. Nach der Beschreibung des Ablaufs zeigt Abbildung 4.1 diesen bildlich dargestellt in einem Sequenzdiagramm.

Ein Migrationsvorgang wird auf dem Quellhost angestoßen. Wie in Abschnitt 4.1 bereits erwähnt, müssen sowohl der Prozess auf dem Zielhost, als auch auf der VM bereits auf den Eingang von Nachrichten warten. Nun schickt der Quellhost zuerst die `prepare_dh`-Nachricht (`msg_id=0`) an den Zielhost. Die in Abschnitt 4.3 beschriebene Datenstruktur ist in dieser Nachricht nur teilweise gefüllt. Der Quellhost muss folgende Datenfelder bereits initialisiert haben:

- `mig_id`
- `msg_id`
- `err_if`
- `vm_iaddr_old`
- `dh_iaddr`
- `vm_maddr`

Die anderen Datenfelder der PDU werden erst im späteren Verlauf des Vorgangs befüllt. Der Zielhost empfängt die `prepare_dh`-Nachricht und generiert dann die IP-Adresse, die der VM im Zielnetz zugeordnet wird. Diese wird in das Datenfeld `vm_iaddr_new` geschrieben. Daraufhin erstellt der Zielhost einen IP-Tunnelendpunkt, um die VM noch vor der tatsächlichen Umsiedlung im Zielnetz erreichbar zu machen. Dabei generiert der Zielhost auch einen Tunnelnamen, der in das Datenfeld `tun_name` geschrieben wird. Sind diese Aktivitäten auf dem Zielhost abgeschlossen, so schickt dieser die `dh_ready`-Nachricht (`msg_id=1`) als Antwort an den Quellhost zurück. Zu diesem Zeitpunkt sind folgende Datenfelder der PDU bereits initialisiert:

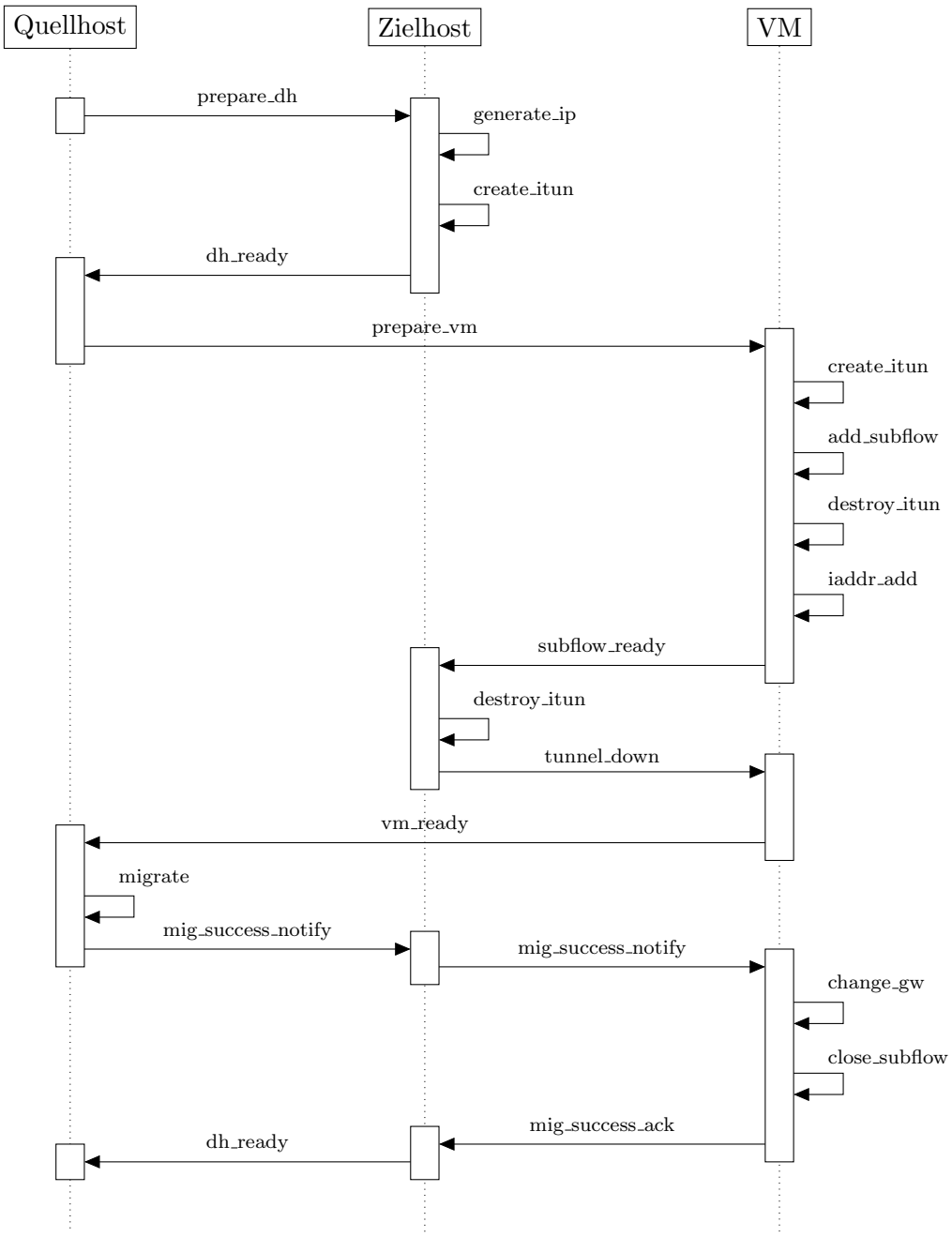

Abbildung 4.1: Sequenzdiagramm des Protokollablaufs

- `mig_id`
- `msg_id`
- `err_if`
- `vm_iaddr_old`
- `vm_iaddr_new`
- `dh_iaddr`
- `vm_maddr`
- `tun_name`

Nachdem der Quellhost vom Zielhost die `dh_ready`-Nachricht empfangen hat, schickt dieser die `prepare_vm`-Nachricht (`msg_id=2`) an die VM. Die VM beginnt nach Erhalt dieser Nachricht mit den Vorbereitungen für eine Migration. Sie erstellt einen IP-Tunnelendpunkt, um einen IP-Tunnel mit dem oben erwähnten IP-Tunnelendpunkt des Zielhosts etablieren zu können. Somit ist die VM zu diesem Zeitpunkt bereits über die IP-Adresse, die ihr im Zielnetz zugeteilt werden wird, erreichbar. Dann folgt das Hinzufügen der Subflows. Das kann lediglich von der Applikation, die Verbindungen zu Clients hat (Server-Applikation), selbst erledigt werden. Das Steuerungsprotokoll stößt diesen Vorgang lediglich an. Von der Server-Applikation erhält das Steuerungsprotokoll nun eine Rückmeldung, wie viele Verbindungen zu Clients die Applikation aktiv besitzt und zu wie vielen dieser Verbindungen ein zusätzlicher Subflow hinzugefügt werden konnte. Diese beiden Informationen speichert die VM in den Datenfeldern `no_clis` und `no_clis_sa`. Zu diesem Zeitpunkt ist die komplette PDU ausgefüllt. Nach dem Hinzufügen der Subflows ist der IP-Tunnel nicht mehr notwendig und wird auf Seiten der VM an dieser Stelle abgebaut. Anschließend wird der Netzwerkschnittstelle die IP-Adresse der VM im Zielnetz hinzugefügt.

Daraufhin benachrichtigt die VM den Zielhost, dass auch dieser seinen IP-Tunnelendpunkt nicht mehr benötigt. Dies geschieht mit der `subflow_ready`-Nachricht (`msg_id=3`). Der Zielhost zerstört seinen IP-Tunnelendpunkt und bestätigt dies der VM durch das Senden der Nachricht `tunnel_down` (`msg_id=4`).

Damit ist die Vorbereitung der VM abgeschlossen und diese schickt dem Quellhost die Nachricht `vm_ready` (`msg_id=5`). Zu diesem Zeitpunkt kann der Quellhost sicher sein, dass sowohl Zielhost, als auch die VM bereit für die Migration sind. Der Quellhost initiiert die tatsächliche Migration.

Nachdem die Migration erfolgreich abgelaufen ist, teilt der Quellhost dies dem Zielhost mit der `mig_success_notify`-Nachricht (`msg_id=6`) mit. Der Zielhost leitet diese Nachricht an die VM weiter. Sobald die VM diese Nachricht vom Zielhost erhält, ändert die VM das Default Gateway. Zusätzlich stößt das Steuerungsprotokoll auf der VM die Server-Applikation an, die Subflows, die nicht mehr genutzt werden, zu zerstören. Dann benachrichtigt die VM den Zielhost mit der `mig_success_ack`-Nachricht (`msg_id=7`), dass die Nachbereitung auf Seiten der VM abgeschlossen werden konnte. Auch diese Nachricht leitet der Zielhost lediglich an den Quellhost weiter. Der Umweg über den Zielhost ist bei den Nachrichten `mig_success_notify` und `mig_success_ack` notwendig, da die VM vor der Änderung des Default Gateways für den Quellhost nicht erreichbar ist.

msg_id	Nachricht
0	prepare_dh
1	dh_ready
2	prepare_vm
3	subflow_ready
4	tunnel_down
5	vm_ready
6	mig_success_notify
7	mig_success_ack

Tabelle 4.1: Zuordnung der Nachrichten-IDs zu den Nachrichten

Mit dem Erhalt der `mig_success_ack`-Nachricht auf dem Quellhost ist der Migrationsvorgang abgeschlossen. Die Instanzen des Steuerungsprotokolls auf dem Zielhost und auf der VM warten auf neue Verbindungsanfragen. Die Instanz auf dem Quellhost terminiert.

Tabelle 4.1 gibt einen Überblick über die Nachrichten des Protokolls und deren zugehörigen `msg_id`s.

4.3 Beschreibung der Protocol Data Unit

Um einen klar definierten Informationsaustausch zwischen Quellhost, Zielhost und VM sicherzustellen, wird hier eine Protocol Data Unit entwickelt, die die benötigten Informationen zur Verfügung stellt. Aus Gründen der Einfachheit wurde eine PDU entwickelt, die zu jedem Zeitpunkt des Ablaufs sicherstellt, dass alle relevanten Daten zur Verfügung stehen.

```
struct pdu{
    int mig_id;
    int msg_id;
    int err_id;
    int vm_iaddr_old[16];
    int vm_iaddr_new[16];
    int dh_iaddr[16];
    char vm_maddr[18];
    char tun_name[64];
    int no_clis;
    int no_clis_sa;
};
```

Abbildung 4.2 zeigt die jeweilige Breite der einzelnen Felder der PDU.

msg_id: Diese Variable ermöglicht eine eindeutige Zuordnung einer empfangenen Nachricht zu einem Migrationsvorgang. Es muss sichergestellt werden, dass diese ID eindeutig ist, dass also nie zwei oder mehrere Migrationsvorgänge die gleiche ID besitzen. Ist dies gewährleistet, so kann das Protokoll doppelt geschickte sowie in falscher Reihenfolge eingegangene Nachrichten erkennen und darauf reagieren.

msg_id: Anhand dieser Variable weiß der Empfänger der Nachricht, in welchem Schritt des Protokollablaufs er sich gerade befindet und demnach auch, was die zu erledigenden Aktionen sind. Zusammen mit der oben beschriebenen `mig_id`-Variable kann ein Duplikat

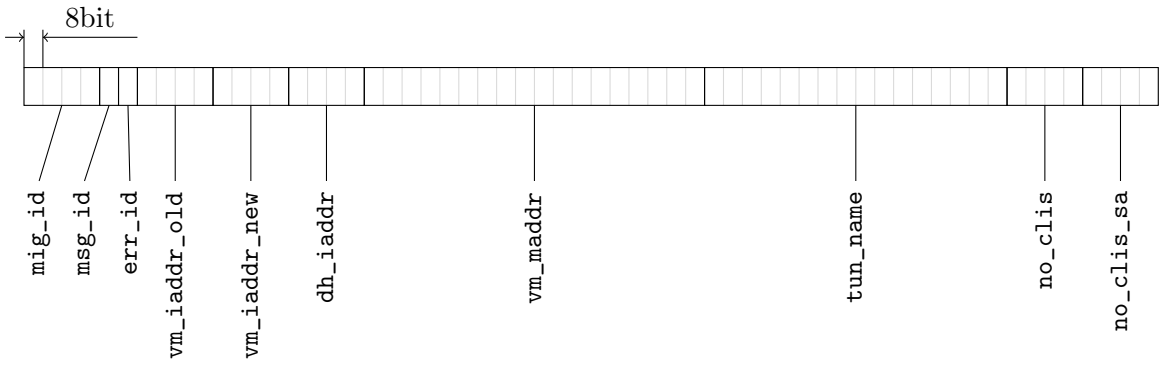

Abbildung 4.2: Breite der PDU-Felder

oder eine Nachricht in der falschen Reihenfolge eindeutig identifiziert werden. Welche Werte `msg_id` annehmen kann und welche Bedeutung diesen Werten zugeschrieben wurde, ist in Abschnitt 4.2 beschrieben.

err_id: Die `err_id`-Variable dient der Beschreibung eines auftretenden Fehlers. Anhand dem Wert der Variable kann nachvollzogen werden, an welcher Stelle im Protokollablauf der Fehler aufgetreten ist und welche Art der Fehlerbehandlung nötig ist, um die Kommunikationspartner in den Ausgangszustand zurückzuversetzen. Bei reibungslosem Ablaufen des Protokolls besitzt die Variable stets den Wert `0`. Tritt ein Fehler auf, so nimmt diese Variable einen Wert zwischen `1` und `7` an. Die Zuordnung der Fehlernummern zu den Fehlercodes wird in Tabelle 4.2 gezeigt.

vm_iaddr_old: Diese Variable enthält die IP-Adresse der VM, die diese vor der Migration besitzt. Der Wert der Variablen ist von Anfang an bekannt und ändert sich im gesamten Protokollablauf nicht.

vm_iaddr_new: Diese Variable enthält die IP-Adresse der VM, die diese nach der Migration besitzen wird. Dieser Wert wird im Protokollverlauf vom Zielhost gesetzt, da der Zielhost die einzige Partei ist, die sich im Zielnetz der Migration befindet. Nachdem diese Variable wie beschrieben gesetzt wurde, ändert sich diese auch nicht mehr.

dh_iaddr: Diese Variable enthält die IP-Adresse des Zielhosts. Diese ist bereits beim Anstoß der Protokollroutine bekannt und ändert sich auch nicht während der Migration.

vm_maddr: Diese Variable enthält die MAC-Adresse der VM, die migriert wird. Das Datenfeld ist nicht zwingend notwendig, wenn allerdings der Zielhost eine IP-Adresse für die VM mittels DHCP generiert, kann diese durchaus von Nutzen sein. Ebenso ist es denkbar, die `mig_id` mithilfe der MAC-Adresse zu generieren.

tun_name: Die Variable `tun_name` beinhaltet den Namen des IP-Tunnels, der im Migrationsprozess notwendig ist. Der Name des Tunnels dient der eindeutigen Identifizierung des Tunnels und wird vom Zielhost und von der VM benötigt, um den Tunnel zu erstellen und wieder zu zerstören.

no_clis: Diese Variable speichert die Anzahl der Clients, mit der die VM eine bestehende Verbindung hat. Zusammen mit der nächsten Variable `no_clis_sa` kann damit ein

prozentualer Anteil errechnet werden, der angibt, wieviele Clients voraussichtlich nach der Migration noch eine Verbindung zur VM haben werden. Anhand dieser Zahl kann dann entschieden werden, ob die Migration fortgeführt, oder ob diese noch vor dem Umziehen der VM auf den Zielhost abgebrochen werden soll.

no_clis_sa: Diese Variable enthält die Anzahl der Clients, zu welchen ein zweiter Subflow von der VM aus aufgebaut werden konnte. Wie bei der vorigen Variable `no_clis` bereits erwähnt, lässt sich aus diesen beiden Variablen ein prozentualer Anteil von Verbindungen, die erhalten werden können errechnen. Aufgrund dessen kann dann entschieden werden, ob die VM tatsächlich migriert, oder der Vorgang abgebrochen werden soll.

4.4 Zustandsmaschinen

In diesem Kapitel werden die Zustände, die das Steuerungsprotokoll auf den beteiligten Maschinen annimmt, betrachtet. Auch wird erläutert, wie ein Zustand erreicht wird, welche Tätigkeiten während eines Zustands abgearbeitet werden und wie er verlassen wird. Zuerst wird dies auf Seiten des Quellhosts betrachtet, dann auf dem Zielhost und abschließend in der VM.

4.4.1 Quellhost

Im Folgenden werden die Zustände, die das Steuerungsprotokoll auf dem Quellhost annehmen kann, erläutert. Das zugehörige Zustandsdiagramm wird in Abbildung 4.3 dargestellt.

try: Diesen Zustand nimmt das Protokoll an, sobald es durch einen Nutzer gestartet wird. Der Quellhost schickt in dieser Phase die Nachricht `prepare_dh (msg_id=0)` an den Zielhost. Daraufhin wartet der Quellhost auf eine Antwort von diesem.

dh_ready received: Nachdem der Quellhost die `dh_ready (msg_id=1)` Nachricht vom Zielhost erhalten hat und die `err_id=0` ist, wechselt der Quellhost in den Zustand dh_ready_received. In diesem Zustand sendet der Quellhost die Nachricht `prepare_vm (msg_id=2)` an die VM. Anschließend wartet der Quellhost auf eine Nachricht der VM.

vm_ready received: Nach dem Erhalt der `vm_ready (msg_id=5)` Nachricht von der VM, überprüft der Quellhost, ob `err_id=0`. Ist das der Fall, so geht der Quellhost in den nächsten Zustand „vm_ready received“ über. In diesem Zustand stößt der Quellhost den eigentlichen Migrationsprozess an.

mig_success received: Nachdem die eigentliche Migration mit Erfolg abgeschlossen wurde, wechselt der Quellhost in den Zustand „mig_success received“. Wenn sich der Quellhost in diesem Zustand befindet, so sendet er die Nachricht `mig_success_notify (msg_id=6)` an den Zielhost. Ab diesem Zeitpunkt wartet der Quellhost auf eine Antwortnachricht vom Zielhost. Trifft diese Antwort ein und ist `err_id=0`, so geht der Quellhost in den finalen Zustand über. Das Steuerungsprotokoll wurde in diesem Fall fehlerfrei abgearbeitet und ist an dieser Stelle am Ende angelangt.

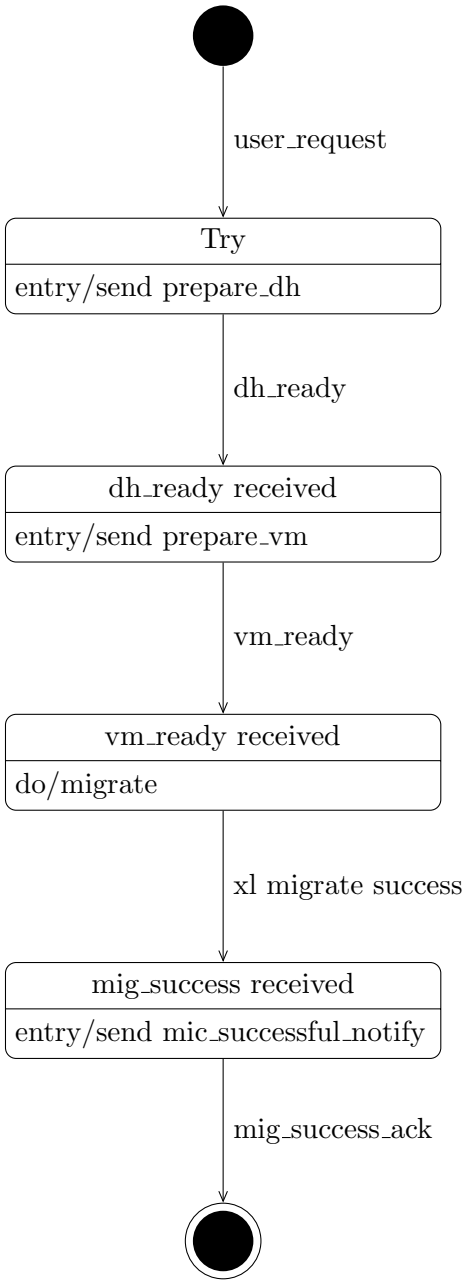

Abbildung 4.3: Zustandsdiagramm Quellhost

4.4.2 Zielhost

Nachfolgend werden die Zustände beschrieben, die der Zielhost während dem Protokollablaufs annehmen kann. Das zugehörige Zustandsdiagramm wird in Abbildung 4.4 dargestellt.

prepare_dh received: Das ist der erste Zustand, in dem sich der Zielhost befindet, nachdem er die `prepare_dh`-Nachricht erhält. In diesem Zustand befindlich, bereitet der Zielhost eine mögliche Migration vor. Er generiert eine IP-Adresse, die der VM nach der Migration zugeteilt wird. Außerdem erstellt der Zielhost einen Tunnelendpunkt, der es später ermöglicht, die VM noch vor der Migration im neuen Zielnetz zu erreichen. Dies wird für das Hinzufügen von Subflows benötigt. Nach Beenden dieser Aktionen sendet der Zielhost eine `dh_ready`-Nachricht an den Quellhost und wartet auf weitere eingehende Nachrichten.

subflow_ready received: Dieser Zustand wird angenommen, wenn die `subflow_ready`-Nachricht zugestellt wurde. Die oben angesprochenen Subflows wurden an dieser Stelle bereits von der VM hinzugefügt und der IP-Tunnel wird nicht mehr benötigt. In diesem Zustand wird der Endpunkt auf Seiten des Zielhosts abgebaut. Ist dies geschehen, so sendet der Zielhost die `tunnel_down`-Nachricht an die VM zurück und wartet auf weitere Nachrichten.

mig_success_notify received: In diesen Zustand wechselt der Zielhost nach dem Erhalt einer `mig_success_notify`-Nachricht. Der Zielhost leitet diese Nachricht lediglich an die VM weiter, Aktionen auf dem Zielhost werden keine durchgeführt. Nach der Weiterleitung wartet der Zielhost auf weitere Nachrichten.

mig_success_ack received: Diesen Zustand nimmt der Zielhost bei Erhalt einer `mig_success_ack`-Nachricht an. Ähnlich wie im Zustand „mig_success_notify received“, leitet der Zielhost auch diese Nachricht nur weiter, in diesem Fall kommt die Nachricht von der VM und wird an den Quellhost weitergeleitet. Danach ist der Migrationsvorgang auf Seiten des Zielhosts beendet. Dieser wechselt in den Endzustand.

4.4.3 Virtuelle Maschine

Nachfolgende Auflistung erläutert die Zustände, die die VM während des Protokollablaufs annehmen kann. Abbildung 4.5 zeigt das zugehörige Zustandsdiagramm.

prepare_vm received: Dies ist der erste Zustand, in den die VM nach dem Erhalt der `prepare_vm`-Nachricht übergeht. In diesem Zustand bereitet sich die VM auf eine mögliche Migration vor. Sie erstellt einen Tunnelendpunkt, um schon vor der Migration über ihre neue IP-Adresse erreichbar zu sein. Das ist notwendig, um die zusätzlichen Subflows zu den aktuell verbundenen Clients zu erstellen. Anschließend fügt die VM die zusätzlichen Subflows den bestehenden Verbindungen zu Clients hinzu. Daraufhin wird der nicht mehr benötigte IP-Tunnelendpunkt abgebaut. Dann stellt die VM sicher, dass sie auch nach der Migration im neuen Netz erreichbar ist, indem die IP-Adresse im Zielnetz der Netzwerkschnittstelle zugeordnet wird. Nachdem diese Aktivitäten abgeschlossen wurden, schickt die VM dem Zielhost die Nachricht `subflow_ready`, um dem Zielhost zu signalisieren, dass auch dieser seinen Tunnelendpunkt nicht mehr benötigt. Anschließend wartet die VM auf weitere Nachrichten.

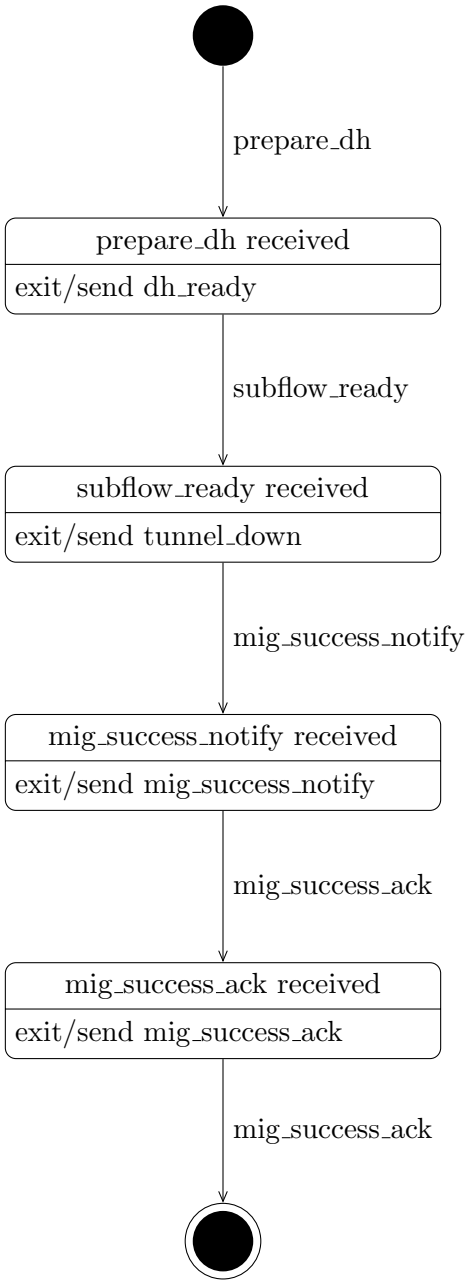

Abbildung 4.4: Zustandsdiagramm Zielhost

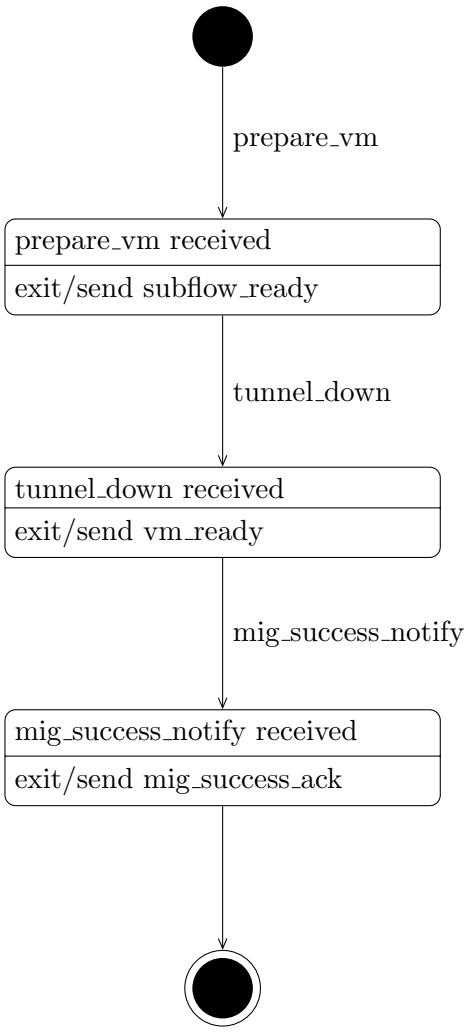

Abbildung 4.5: Zustandsdiagramm VM

err_id	Fehlercode
1	EDHPREP
2	EVMPREP
3	EDESTROYTUN
4	EMIG
5	EFOLLOWUPVM
6	EFOLLOWUPDH
7	EOUTOFORDER

Tabelle 4.2: Zuordnung der Fehlernummern zu den Fehlercodes

tunnel_down received: Mit dem Erhalt der Nachricht `tunnel_down` vom Zielhost, geht die VM in den „tunnel_down received“-Zustand über. Daraufhin schickt die VM die Nachricht `vm_ready` an den Quellhost. Weitere Aktivitäten finden in diesem Zustand nicht statt, die VM wartet auf weitere Nachrichten.

mig_success_notify received: Diesen Zustand nimmt die VM an, wenn sie die `mig_success_notify`-Nachricht erhalten hat. Daraufhin ändert die VM das Default Gateway, um ausgehenden Datenverkehr nicht mehr über das Gateway vor der Migration zu senden, sondern über das Gateway auf dem Zielhost. Dies stellt die Erreichbarkeit der VM im Internet sicher. Außerdem werden in diesem Zustand auch die nicht mehr benötigten Subflows von vor der Migration zerstört. Sind diese Schritte abgeschlossen, so schickt die VM die `mig_success_ack`-Nachricht an den Zielhost zurück und geht in den Endzustand über. Für die VM ist der Protokollablauf an dieser Stelle beendet.

4.5 Fehlerbehandlung

In Abschnitt 2.2 wurden Anforderungen an das Protokoll gestellt. Eine funktionale Anforderung war die Fehlererkennung und -rehabilitation. Das bedeutet, das Protokoll muss das Auftreten eines Fehlers erkennen und in der Lage sein, alle Protokollpartner in einen bereinigten Ausgangszustand zurückzuversetzten. Darüber hinaus war eine weitere nicht-funktionale Anforderung die Robustheit des Protokolls. Um eine gewisse Robustheit erreichen zu können, darf keiner der Kommunikationspartei den Protokollablauf bei Auftreten eines Fehlers blockieren. Die Funktionalität des Protokolls darf von einem Fehler nicht beeinträchtigt werden. Dieses Kapitel erklärt die Fehlerbehandlung des Protokolls.

Es gilt zwischen zwei unterschiedlichen Fehlerfällen zu unterscheiden. Zum Einen, wenn ein Fehler bei einer lokalen Aktion von einem der drei Partner auftritt, zum Anderen, wenn eine Nachricht bei der Übertragung verloren geht, doppelt geschickt wird, oder in der falschen Reihenfolge ankommt.

Der Umgang mit Fehlern aus der ersten Kategorie ist einfacher, da diese leichter zu erkennen sind und die beiden anderen Kommunikationspartner informiert werden und auf den Fehler reagieren können. Zu diesem Zweck wurden in der in Abschnitt 4.3 gezeigten PDU `err_ids` eingeführt. Tabelle 4.2 zeigt mögliche Fehlernummern und ihre Zuordnung zu den Fehlercodes.

Abbildung 4.6 zeigt das Zustandsdiagramm des Quellhosts aus Unterabschnitt 4.4.1, an dieser Stelle aber mit Fehlerbehandlung. Der Quellhost koordiniert die Fehlerbehandlung, das bedeutet, beim Auftreten eines Fehlers wird dieser immer an den Quellhost gemeldet.

Aus der Grafik wird klar, welche Fehler zu welchem Zeitpunkt auftreten können. Der Zielhost und die VM reagieren immer gleich auf das Auftreten von Fehlern. Sie versuchen ihren Ausgangszustand herzustellen und melden den aufgetretenen Fehler an den Quellhost weiter.

Nachrichten in falscher Reihenfolge werden an dieser Stelle nicht berücksichtigt. Diese Fehlerart wird später behandelt. Im folgenden Abschnitt wird der Umgang mit den auftretenden Fehlern im Detail beschrieben.

EDHPREP: Wenn die `err_id=1` ist, dann ist ein Fehler bei der Vorbereitung des Zielhosts aufgetreten. Der Zielhost versetzt sich bei Auftreten des Fehlers selbst wieder in einen bereinigten Zustand und meldet diesen Fehler in seiner Antwort an den Quellhost. Dieser weiß dann, dass ein Fehler aufgetreten ist und bricht den Migrationsvorgang ab. Ein weiteres Bereinigen ist nicht notwendig, da die VM zu diesem Zeitpunkt noch nicht verändert wurde und somit auch nichts rückgängig gemacht werden muss.

EVMPREP: Wenn `err_id=2`, dann konnte die Vorbereitung der VM nicht ohne Fehler abgeschlossen werden. Die VM bereinigt getätigte Änderungen selbstständig und meldet diesen Fehler an den Quellhost zurück. Dieser weiß, dass zu diesem Zeitpunkt der Zielhost bereits vorbereitet wurde. Der Quellhost schickt eine Aufforderung an den Zielhost, die Vorbereitungen wieder rückgängig zu machen. Ist dies geschehen, so beendet der Quellhost den Migrationsvorgang.

EDESTROYTUN: `err_id=3` wird verwendet, um eine missglückte Zerstörung des IP-Tunnelendpunkts auf der Zielhost-Seite zu erkennen. Der Zielhost kann nichts weiter machen, als den Fehler an die VM zu melden. Die VM bereinigt dann ihrerseits getätigte Vorbereitungen und meldet den Fehler weiter an den Quellhost. Daraufhin benachrichtigt wiederum der Quellhost den Zielhost mit der Aufforderung, die Vorbereitungen, die der Zielhost bereits getätigt hat, rückgängig zu machen. Ein Teil dieser Bereinigung besteht allerdings darin, diesen IP-Tunnelendpunkt zu zerstören, der diesen Fehler initial ausgelöst hat. Möglicherweise ist es auch in diesem Schritt nicht möglich, den IP-Tunnelendpunkt zu zerstören. Höchstwahrscheinlich ist dann der IP-Tunnelendpunkt gar nicht mehr existent. Ist er jedoch noch vorhanden und scheitert der Zielhost auch in dem Bereinigungsschritt den IP-Tunnelendpunkt abzubauen, so ist ein händisches Eingreifen nicht zu vermeiden. Nachdem alle möglichen Bereinigungen durchgeführt wurden, beendet der Quellhost den Migrationsvorgang.

EMIG: Wenn `err_id=4`, dann konnte die eigentliche Migration der VM auf den Zielhost nicht erfolgreich durchgeführt werden. In diesem Fall muss sowohl der Zielhost als auch die VM aufgefordert werden, den Ausganszustand wiederherzustellen. Nach dem Erhalt einer Bestätigung der beiden Kommunikationspartner beendet der Quellhost dann den aktuellen Migrationsvorgang.

EFOLLOWUPVM: Wenn `err_id=5`, dann konnte die Nachbereitung der VM nicht erfolgreich abgeschlossen werden. Zu diesem Zeitpunkt läuft die VM schon auf dem Zielhost. Der Quellhost fordert den Zielhost auf, die VM auf den Quellhost zurückzumigrieren. Nachdem die Migration zurück auf den Quellhost abgeschlossen werden konnte, muss dieser den Zielhost und die VM anstoßen, die vorhergegangenen Änderungen wieder rückgängig zu machen. Danach beendet der Quellhost diesen Migrationsvorgang und meldet dem Nutzer die fehlgeschlagene Migration.

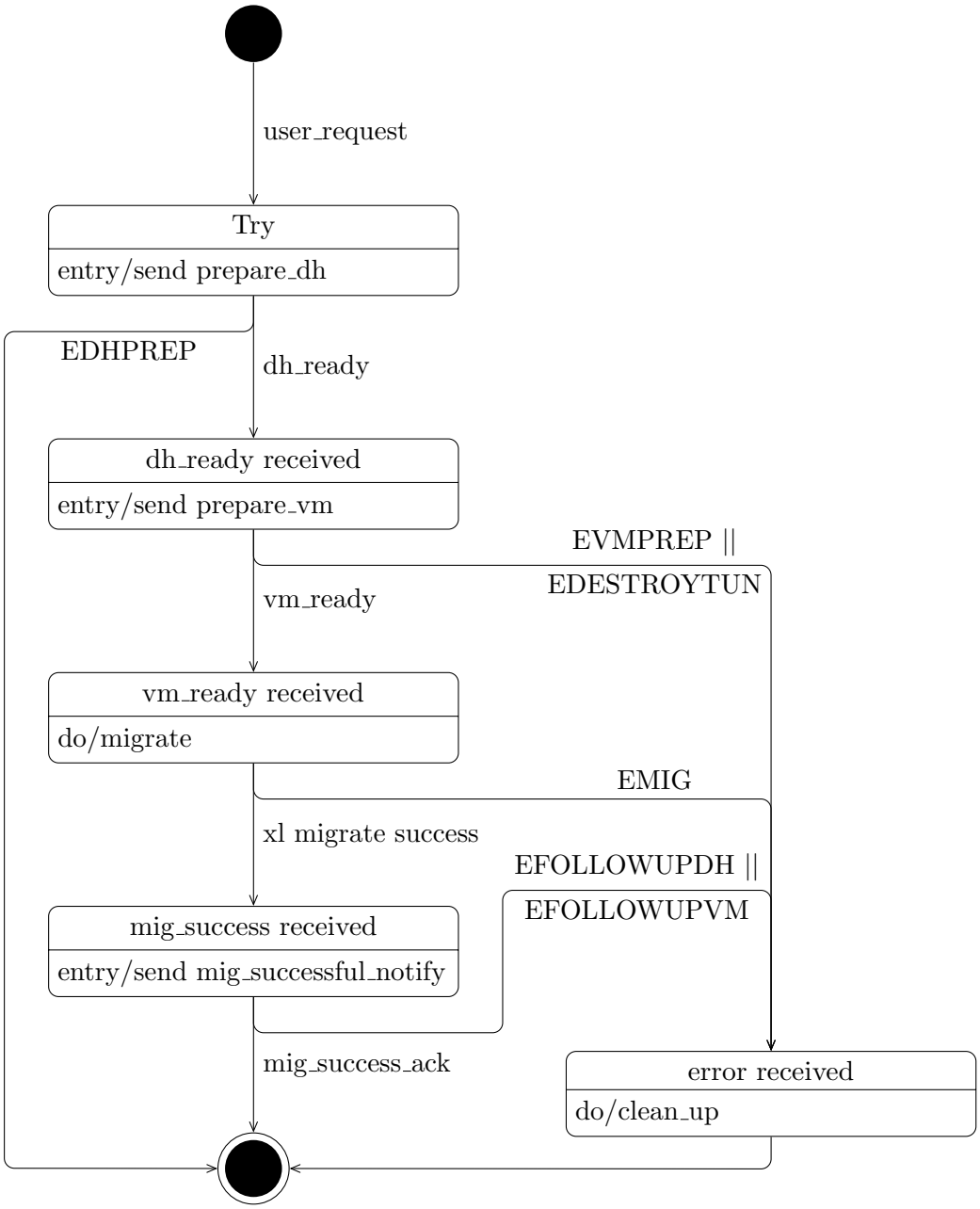

Abbildung 4.6: Zustandsdiagramm Quellhost mit Fehlerbehandlung

	Quellhost	Zielhost	VM
EDHPREP	E	S	∅
EVMPREP	S+E	E	S
EDESTROYTUN	S+E	S+E	S+E
EMIG	S	E	E
EFOLLOWUPVM	S+E	S+E	S+E
EFOLLOWUPDH	S+E	S+E	E

Tabelle 4.3: Übersicht über Sender und Empfänger von Fehlercodes

EFOLLOWUPDH: Beim Auftreten von `err_id=6` ist ein Fehler bei der Nachbereitung der Migration auf dem Zielhost aufgetreten. Die Fehlerbehandlung beim Auftreten dieses Fehlers unterscheidet sich nicht von der Fehlerbehandlung von `err_id=EFOLLOWUPVM`. Da der Zielhost die an dieser Stelle des Protokolls gesendete Nachricht nur durchreicht und keine lokalen Operationen tätigt, sollte dieser Fehler nicht auftreten, außer die VM antwortet nicht auf die `mig_success_notify`-Nachricht.

Tritt ein Fehler bei der Datenübertragung auf, so äussert sich dies entweder im Empfang von Duplikaten oder Nachrichten in der falschen Reihenfolge oder es wird keine Nachricht mehr empfangen. Werden Duplikate oder Nachrichten in der falschen Reihenfolge empfangen, so kann dies durch die `msg_id` detektiert werden. Dann kann der Sender dieser falschen Nachricht mit der `err_id=EOUTOFORDER` über den Fehler benachrichtigt werden. Wenn allerdings der Kommunikationspartner komplett vom Netz abgeschnitten ist, dann muss die andere Partei nach einer bestimmten Zeit aufhören auf eine Antwort zu warten. Nach diesem Timeout müssen alle Kommunikationspartner den Ursprungszustand wieder herstellen. Die vom Netz abgeschnittene Komponente kann lediglich den Initialzustand wiederherstellen. Sie ist allerdings nicht in der Lage, weitere Instanzen zu benachrichtigen, dass der Migrationsvorgang fehlgeschlagen ist. Das muss demnach die Komponente machen, die für länger als dem Timeout keine Antwort von dem abgekapseltem Partner mehr bekommen hat. Dieser Kommunikationspartner muss sich darum kümmern, dass der Bereinigungsprozess auch auf der dritten verbleibenden Komponente stattfindet. Wenn beispielsweise die VM auf eine `prepare_vm`-Nachricht nicht innerhalb eines fest definierten Zeitintervalls antwortet, so muss der Quellhost auch den Zielhost benachrichtigen, dass dieser seinen Ursprungszustand wiederherstellt.

Tabelle 4.3 zeigt eine Übersicht, welche Kommunikationspartei welche Fehlercodes sendet und empfängt. Das Kürzel „S“ steht dabei für „sendet“, „E“ für „empfängt“ und „S+E“ steht für „sendet und empfängt“. Wird ein Fehlercode weder gesendet, noch empfangen, so wird die mit „∅“ gekennzeichnet.

5 Implementierung

Dieses Kapitel beschreibt die in der Arbeit entwickelte Implementierung des in Kapitel 4 entwickelten Protokolls. Anfangs wird auf die verwendete Laborumgebung eingegangen. Daraufhin folgt eine Aufzählung der verwendeten Programme und Protokolle, zusammen mit einer kurzen Erklärung dieser. Anschließend wird das entwickelte Programm vorgestellt. Dieses besteht aus drei Teilen, die jeweils auf den drei beteiligten Parteien, dem Quellhost, dem Zielhost und auf der VM, laufen.

5.1 Beschreibung der Laborumgebung

Dieser Abschnitt beschreibt die Laborumgebung, in der die Implementierung zu Testzwecken ausgerollt wurde. Diese Laborumgebung versucht das in Abschnitt 2.1 dargestellte Szenario nachzubilden. Es werden zwei Hosts benötigt, um die Implementierung zu realisieren. Zum Einen der Quellhost, auf dem die VM zu Beginn der Protokollausführung läuft, zum Anderen der Zielhost, auf den die VM migriert wird. Um sicherzustellen, dass die Migration verbindungserhaltend abläuft, benötigt man außerdem mindestens einen Client, der bereits vor der Migration eine bestehende Verbindung zur VM besitzt. Dieser Client ist in der hier vorgestellten Laborumgebung ebenfalls auf dem Quellhost. Diese virtuelle Umgebung wird schematisch in Abbildung 5.1 dargestellt. Um anzudeuten, dass die VM migriert wird, wurde der gestrichelte Pfeil und die gestrichelte VM auf dem Zielhost mit abgebildet. Die VM läuft zu keinem Zeitpunkt aktiv auf beiden Hosts.

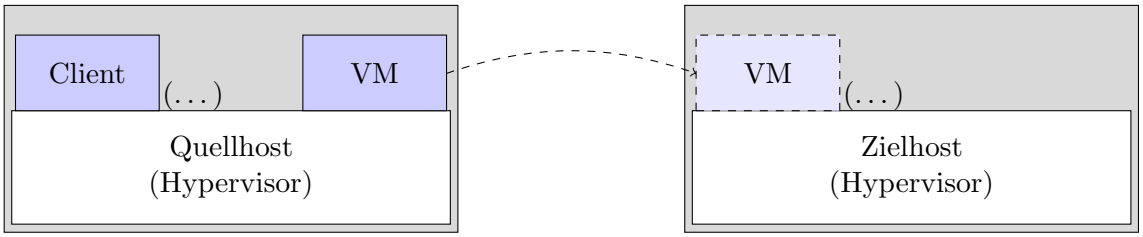

Abbildung 5.1: Virtuelle Umgebung

Auf den beiden Hosts ist Debian mit der Kernelversion 3.16 installiert ([deb18]). Als Hypervisor agiert ebenfalls auf beiden Maschinen Xen in der Version 4.4.1. ([xen18]). Der verwendete Toolstack ist `xl` ([xl17]).

Auf dem Client und auf der VM, die migriert wird, ist ebenfalls ein Debian System installiert. Allerdings wird hier ein selbst kompilierter Kernel der Version 4.4.70 verwendet. Dieser implementiert das in Abschnitt 3.3 vorgestellte Protokoll MPTCP. Genauere Informationen zu diesem Kernel finden sich im nachfolgenden Abschnitt 5.2.

Nun werfen wir einen Blick auf die Netztopologie, die von der Laborumgebung realisiert wird. Der Quellhost und der Zielhost befinden sich beide im `192.168.0.0/24` Netz und können darüber miteinander kommunizieren. Der Quellhost ist zusätzlich im `192.168.1.0/24` Netz.

Das ist auch das Netz, in dem sich sowohl der Client, als auch die VM vor der Migration befindet. Der Zielhost befindet sich neben dem `192.168.0.0/24` Netz noch im `192.168.2.0/24` Netz. Das ist das Netz, in dem sich die VM nach der Migration befinden wird. Die beschriebene Netztopologie ist in Abbildung 5.2 dargestellt.

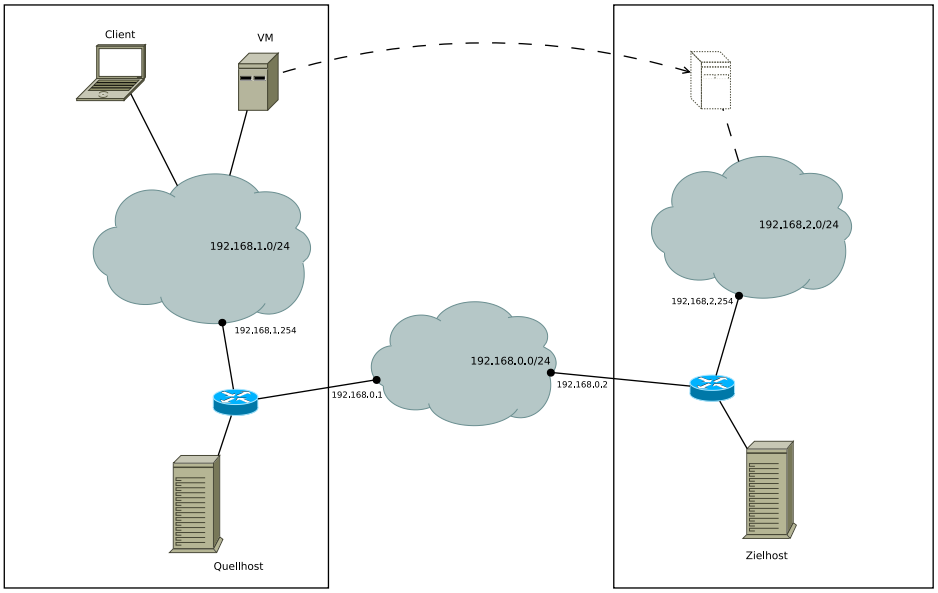

Abbildung 5.2: Skizze Laborumgebung

Um sicherzustellen, dass die VM nach der Migration für den Client erreichbar ist, muss eine Route vom Client in das `192.168.2.0/24` Netz existieren. Genauso muss für die VM nach der Migration im `192.168.2.0/24` Netz eine Route in das `192.168.1.0/24` Netz existieren. Die beiden Maschinen Quellhost und Zielhost agieren in der Laborumgebung als Router. Der Quellhost weiß, dass er Netzwerkverkehr in das `192.168.2.0/24` Netz an die `192.168.0.2` über die eth1 Schnittstelle routen muss. Analog dazu weiß der Zielhost, dass er Verkehr in das `192.168.1.0/24` Netz über `192.168.0.1` und über die lstinlineeth1-Schnittstelle routen muss. Darüberhinaus gibt es noch die Routen der Hosts, die das Netz für die virtuellen Maschinen adressieren. Diese nutzen ein anderes Netzwerkinterface, in der verwendeten Laborumgebung das „br_man“-Interface. Die Routingtabellen von Client und VM müssen im Normalfall nicht abgeändert werden. Die verwendete Laborumgebung erfordert das Löschen des Routingeintrags in das eigene Netz auf der VM. Vor der Migration befindet sich die VM im `192.168.1.0/24` Netz. Es existiert ein Routingeintrag der Netzverkehr in das eigene Netz nicht routet, sondern dafür sorgt, dass die Zieladresse mit Methoden der Schicht 2 des OSI-Schichtenmodells ermittelt wird. Wird die VM in das `192.168.2.0/24` Netz migriert und dieser Routingeintrag existiert, so kann das `192.168.1.0/24` Netz nicht mehr erreicht werden. Würde sich der Client allerdings in einem anderen Netz befinden, so wäre diese Änderung nicht notwendig. Abbildung 5.3 zeigt die relevanten Auszüge aus den Routingtabellen von Quellhost, Zielhost VM und Client.

```
Quellhost:
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 br_man
192.168.2.0     192.168.0.2     255.255.255.0   UG    0      0        0 eth1

Zielhost:
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
192.168.1.0     192.168.0.1     255.255.255.0   UG    0      0        0 eth1
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 br_man

VM:
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.1.254   0.0.0.0         UG    0      0        0 eth0

Client:
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.1.254   0.0.0.0         UG    0      0        0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

Abbildung 5.3: Auszug aus den Routingtabellen

5.2 Verwendung bereits existierender Programme und Protokolle

Dieser Abschnitt fasst die von der Implementierung verwendeten Programme und Protokolle zusammen. Beginnend wird der Xen Hypervisor kurz beschrieben. Anschließend wird die Verwendung des Programms NBD erläutert. Danach werfen wir einen Blick auf die Linux Implementierung von MPTCP und auf die „Enhanced Socket-API for MPTCP“.

5.2.1 Xen Hypervisor

Die in dieser Arbeit entwickelte Implementierung verwendet Xen als Hypervisor, um virtuelle Maschinen zu hosten. Ein Hypervisor ist eine Schicht zwischen Hardware und mehreren virtuellen Maschinen, die auf dem gleichen Host laufen. Die virtuellen Maschinen, die auf einem Xen-Host laufen, werden „Domains“ genannt. Die erste Domain, „Dom0“ hat besondere Rechte, beispielsweise kann diese direkt auf die Hardware zugreifen. Außerdem können mit der Dom0 die anderen virtuellen Maschinen gesteuert werden. Xen kann ohne das Starten der Dom0 nicht benutzt werden. Die Dom0 ist die erste VM, die von Xen gestartet wird (vgl. [xen18]).

Die Schnittstelle, mit der die Dom0 die anderen VMs steuert, ist der sogenannte „Toolstack“. Es existieren verschiedene Toolstacks. Der in dieser Arbeit gewählte Toolstack ist der „XL-Toolstack“[xl17]. Dieser wird vom Team des Xen Projects entwickelt und ist der Standard-Toolstack. Mit diesem können die VMs unter anderem gestartet, gestoppt, zerstört und auch migriert werden.

5.2.2 Network Block Device

Bei der Live-Migration virtueller Maschinen, wie sie bereits implementiert ist und in dieser Arbeit auch verwendet wird, wird nicht das komplette Festplattenimage auf den neuen Host

kopiert. Es wird lediglich der Arbeitsspeicher und im letzten Schritt auch der Zustand der CPU kopiert. Genauere Informationen dazu findet man in Abschnitt 3.1, sowie in der von Clark et al. veröffentlichten Arbeit [CFH+05]. Die VM muss aber auch nach der Migration auf ihr Image zugreifen können. Daher ist es notwendig, dass das Image der VM als „shared storage“ über das Netzwerk zur Verfügung steht. In der in dieser Arbeit verwendeten Laborumgebung ist das Image der zu migrierenden VM auf dem Quellhost. Um das Image für den Zielhost verfügbar zu machen, wird das in Linux implementierte Protokoll NBD verwendet. Dieses ermöglicht es, eine virtuelle Festplatte über eine TCP-Verbindung zur Verfügung zu stellen. Dafür muss auf dem Quellhost sowohl das Programm „nbd-server“, als auch „nbd-client“ installiert sein. Auf dem Zielhost ist es ausreichend nur „nbd-client“ zu installieren. Der Server-Dienst stellt die Image-Datei der VM dann sowohl dem NBD-Client auf dem Quellhost, als auch auf dem Zielhost zur Verfügung (vgl. [Verb][Vera][nbd18]).

5.2.3 MPTCP Implementierung

Das in Abschnitt 3.3 beschriebene Protokoll wird von der Laborumgebung auf dem Client und auf der VM eingesetzt. Es existiert eine Implementierung des Protokolls für einige Linux-Kernel. Der in der Laborumgebung verwendete Kernel ist ein Linux-Kernel der Version 4.4.70. Die verwendete MPTCP Version ist `v0.92`, die zu diesem Zeitpunkt des Kernelbaus aktuellste Version. Der Kernel wurde von der MPTCP-Webseite [mpt18b] heruntergeladen. Anschließend wurde dieser konfiguriert und gebaut.

Bei der Konfiguration ist darauf zu achten, dass zwar die Rubrik „MPTCP: advanced path-manager control“ aktiviert wird, beim Pathmanager allerdings die Standardeinstellungen nicht geändert werden. Der Default Pathmanager kümmert sich nicht um das Hinzufügen oder Abbauen von Subflows. Da wir uns mit dem Steuerungsprotokoll allerdings selbst darum kümmern, wäre es eher hinderlich, diese Aufgabe zusätzlich dem Kernel aufzutragen. Die vollständige Konfiguration, die in der Laborumgebung für die Kernel von Client und VM getätigt wurden, finden sich in der nachfolgenden Aufzählung.

- Aktivieren von `Networking support->Networking options->TCP/IP networking ->MPTCP protocol`
- Aktivieren von `Networking support->Networking options ->TCP: advanced congestion control->MPTCP Linked Increase` und als Standard-Wert setzen
- Aktivieren von `Networking support->Networking options->TCP/IP networking ->MPTCP protocol->MPTCP: advanced path-manager control`
  Beachte: hier keinen der zur Verfügung stehenden Pathmanager auswählen, sondern die vorgefundenen Einstellungen übernehmen.
- Aktivieren von `Networking support->Networking options->IP: advanced router ->IP: policy routing`

Nachdem diese Einstellungen bezüglich MPTCP getätigt wurden, kann der Kernel auf dem Client kompiliert, installiert und gestartet werden. Die VM benötigt noch eine API zur Steuerung der Subflows. Diese muss noch vor dem Bauen des Kernels gepatcht werden. Die Vorgehensweise beschreibt der nächste Abschnitt.

5.2.4 Enhanced Socket-API for MPTCP

Um in der Lage zu sein die Subflows zu steuern, wird eine bereits implementierte API verwendet (vgl [HB16]). Diese „enhanced Socket-API for Multipath TCP“ muss manuell in die Kerneldateien der VM eingepflegt werden. Der Quellcode wurde der MPTCP-Mailingliste ([mpt18a]) entnommen. Aufgrund der fehlenden Informationen, mit welcher MPTCP-Version die API entwickelt wurde, beziehungsweise mit welcher Version die API läuft, wurde die zum Zeitpunkt der Kernelerstellung neueste Version von MPTCP verwendet. Diese ist, wie bereits in Unterabschnitt 5.2.3 erwähnt, `v0.92`. Die API funktioniert mit dieser Version von MPTCP. Nachdem die API gepatcht wurde, kann auch auf der VM der Kernel gebaut werden. Unter Debian kann der Kernel mit dem Befehl `make deb-pkg` gebaut werden. Dies erzeugt unter Anderem ein linux-libc Paket, das es ermöglicht die gepatchte API auch zu verwenden. Das Installieren dieses Pakets erzeugt die C-Standardbibliothek mit den erweiterten Funktionen der Socket-API. Nun kann also der Kernel und die erstellte C-Standardbibliothek installiert und gestartet werden.

Sowohl Client, als auch die VM sind nun in der Lage über MPTCP zu kommunizieren. Die VM ist mittels gepatchter API zusätzlich in der Lage, Subflows zu steuern.

Mithilfe der Socket-API ist es nun möglich die Funktionsweise von MPTCP, insbesondere Subflows einer TCP-Verbindung zu steuern. Tabelle 5.1 zeigt den Funktionsumfang der API.

Name	Input	Output	Beschreibung
MPTCP_GET_SUB_IDS	-	subflow list	Liste der aktuellen Subflows
MPTCP_GET_SUB_TUPLE	id	sub tuple	IP und Port Paar des Subflows id
MPTCP_OPEN_SUB_TUPLE	tuple	-	neuen Subflow mit IP und Port Paar anfragen
MPTCP_CLOSE_SUB_ID	id	-	Subflow `id` schließen
MPTCP_SUB_GETSOCKOPT	id, sock opt	sock ret	leitet die `getsockopt` an den Subflow `id` um und gibt den Rückgabewert der Aktion zurück
MPTCP_SUB_SETSOCKOPT	id, sock opt	-	leitet die `setsockopt` an den Subflow mit `id` weiter

Tabelle 5.1: Funktionsumfang enhanced Socket-API, Quelle: [HB16, S. 4, übersetzt]

Die relevanten Funktionen in dem von der Arbeit behandelten Szenario werden die MPTCP_OPEN_SUB_TUPLE und MPTCP_CLOSE_SUB_ID sein. Damit ist es möglich neue Subflows aufzubauen und nicht mehr benötigte Subflows wieder abzubauen. Außerdem wird zu informativen Zwecken die Funktion MPTCP_GET_SUB_IDS verwendet werden.

5.3 Programmstruktur auf Quellhost

In diesem und den beiden folgenden Abschnitten wird die konkrete Implementierung des Protokolls erklärt. Das Protokoll wurde in C implementiert und gliedert sich in drei Teile. Der erste Teil läuft auf dem Quellhost und wird in diesem Kapitel beschrieben. Den zweiten Teil bildet die Implementierung auf dem Zielhost im nächsten Kapitel und darauf folgt mit der Implementierung des Protokolls auf der VM der dritte Teil.

Der Quellhost koordiniert die Migration. Der Protokollablauf beginnt mit dem Anstoß durch einen Administrator auf dem Quellhost. Diesem werden die IP-Adresse des Zielhost, die IP-Adresse der VM und der Name der VM als Kommandozeilenparameter mit übergeben.

Nach dem Anstoß generiert der Quellhost eine Migrations-ID und ermittelt die MAC-Adresse der zu migrierenden VM. Alle bis dahin zur Verfügung stehenden Daten (IP des Zielhost, IP der VM, Name der VM, MAC-Adresse der VM und Migrations-ID) werden in ein Struct geschrieben. Dieses Struct besitzt die Einträge, die in Abschnitt 4.3 beschrieben wurden. Der Eintrag des Structs `mig_id` wird an dieser Stelle des Protokollablaufs mit `prepare_dh` befüllt und das Struct wird an den Zielhost geschickt. Dann wartet der Quellhost auf eine Antwort vom Zielhost. Wenn diese eingetroffen ist, prüft der Quellhost, ob die Vorbereitung auf dem Zielhost erfolgreich war. Dies geschieht mittels der `msg_id`, die den Wert `dh_ready` besitzen muss und mittels `err_id`, die gleich `0` sein muss. Ist eine dieser Bedingungen verletzt, so beendet der Quellhost den Protokollablauf.

Sind diese Bedingungen allerdings erfüllt, so geht der Quellhost dazu über, die VM vorzubereiten. Dazu setzt er die `msg_id=prepare_vm` und sendet diese Nachricht mit der restlichen Datenstruktur an die VM. Dann wartet er auf eine Antwort von der VM. Diese wertet der Quellhost wieder aus, indem er prüft, ob `msg_id=vm_ready` und `err_id=0` ist. Ist dies nicht der Fall, so muss der im vorigen Schritt vorbereitete Zielhost noch in den Ausgangszustand versetzt werden, bevor der Quellhost den Protokollablauf beenden kann.

Sind alle Bedingungen erfüllt, so geht der Quellhost von einer erfolgreichen Vorbereitung der VM aus und leitet den nächsten Schritt in die Wege. In diesem Schritt findet die tatsächliche Migration statt. Dazu wird die VM mithilfe des in Unterabschnitt 5.2.1 vorgestellten XL-Toolstacks migriert. Der konkrete Befehl findet sich in Listing 5.1.

```
xl migrate <VM_Name> <DestinationIP>
```

Listing 5.1: Migration der VM mit XL-Toolstack

Der Quellhost wartet dann auf den Rückgabewert des Befehls. Ist dieser in Ordnung, so wird zum nächsten Schritt übergegangen. Andernfalls muss die VM und der Zielhost benachrichtigt werden, dass die Migration fehlgeschlagen ist. Die VM befindet sich hierbei unverändert auf dem Quellhost.

Ist die Migration hingegen geglückt, so ändert der Quellhost die `msg_id` auf `mig_success_notify` und sendet diese Nachricht an den Zielhost. Dann wartet der Quellhost wieder auf eine Antwort von diesem. Die Antwort wird dann, wie in den Schritten davor wieder auf Korrektheit überprüft. Wenn `msg_id=mig_success_ack` und `err_id=0`, dann signalisiert der Zielhost dem Quellhost ein erfolgreiches Nachbereiten und der Protokollablauf wird vom Quellhost beendet. Die verbindungserhaltende Migration ist in diesem Fall geglückt. Ist die Antwort vom Zielhost aber nicht in Ordnung, so muss ein Fehler in der Nachbereitung aufgetreten sein. Dann veranlasst der Quellhost das Migrieren der VM vom Zielhost zurück auf den Quellhost und fordert anschließend die VM und den Zielhost auf, den Ausgangszustand wieder herzustellen. Nachdem diese Aktionen getätigt wurden, beendet der Quellhost auch in diesem Fall den Protokollablauf, allerdings nicht mit einer erfolgreichen Migration.

5.4 Programmstruktur auf dem Zielhost

Der Zielhost implementiert einen „Concurrent Server“. Im Gegensatz zum iterativen Server, ist ein konkurrenter Server in der Lage, mehrere Anfragen gleichzeitig zu bedienen. Sobald eine neue Verbindungsanfrage eingeht, startet der Zielhost einen neuen Prozess und akzeptiert die Verbindung zu dem neuen „Client“. Damit ist es möglich, dass der Zielhost mehrere VMs gleichzeitig empfangen kann. Allerdings muss die Protokollimplementierung auf dem Zielhost

dann sicherstellen, dass eine empfangene Nachricht entweder zu einem neuen Migrationsprozess gehört, oder zu einem bereits vorhandenem Migrationsprozess mit `mig_id` gehört und in der richtigen Reihenfolge angekommen ist. Dazu nutzt der Zielhost eine Liste von existierenden Migrationsvorgängen, in der der Fortschritt gespeichert wird. Erhält der Zielhost eine Anfrage mit einer unbekannten `mig_id`, so wird diese mit dem aktuellen Fortschritt in der Liste gespeichert. Existiert die `mig_id` bereits, so überprüft der Zielhost, ob die empfangene Nachricht zum Fortschritt des Migrationsprozesses passt. Dies geschieht anhand der `msg_id`. Wird die richtige Nachricht empfangen, so wird die `msg_id` in der Prozessliste auf den aktuellen Wert, den die `msg_id` in der empfangenen Nachricht besitzt, gesetzt. Sobald der Zielhost die letzte Nachricht eines Migrationsvorgangs abschickt, und der Vorgang für den Zielhost beendet ist, löscht dieser den Migrationsvorgang auch aus der Liste der aktuell stattfindenden Migrationen.

Die erste Nachricht, die der Zielhost in einem Migrationsvorgang erhält, ist die `prepare_dh` Nachricht. Mit dieser Nachricht wird der Zielhost aufgefordert, den Erhalt einer VM vorzubereiten. Mit dem Empfang der Nachricht überprüft der Zielhost, ob eine Migration mit gleicher ID bereits existiert und wenn nicht, so wird die Migrations-ID mit Nachrichten-ID in der Liste der aktuellen Migrationsvorgänge gespeichert. Existiert die Migrations-ID bereits in angesprochener Liste, so wird eine Nachricht mit `err_id=EOUTOFORDER` an die VM zurückgeschickt. Ist die Nachricht in Ordnung, so generiert der Zielhost zuerst eine IP-Adresse, die die VM im Zielnetz erhalten soll. Anschließend wird ein IP-Tunnelname erzeugt und ein Tunnelendpunkt mit diesem Namen aufgebaut. Können diese Aufgaben erledigt werden, so schickt der Zielhost die `dh_ready` Nachricht an den Quellhost zurück. Schlägt eine der erläuterten Aktionen fehl, so bereinigt der Zielhost alle Änderungen bis zu diesem Zeitpunkt und antwortet dem Quellhost mit der `err_id=EDHPREP`. In diesem Fall muss die Migrations-ID in der Liste der aktuell stattfindenden Migrationsvorgängen gelöscht werden. Dann geht der Zielhost wieder in einen wartenden Zustand über.

Erhält der Zielhost eine Nachricht mit `msg_id=subflow_ready`, so stammt diese Nachricht von der VM und benachrichtigt den Zielhost, dass dieser seinen IP-Tunnelendpunkt wieder abbauen kann. Wieder überprüft der Zielhost, ob es den Migrationsvorgang mit der `mig_id` aus der empfangenen Nachricht bereits gibt. Ist das nicht der Fall, oder indiziert die `msg_id` eine falsche Reihenfolge der Nachricht, so antwortet der Zielhost der VM mit einer Nachricht mit dem Inhalt `err_id=EOUTOFORDER`. Andernfalls baut der Zielhost den Tunnelendpunkt ab und antwortet der VM je nach Erfolg entweder mit `msg_id=tunnel_down` oder `err_id=EDESTROYTUN`. Dann geht der Zielhost wieder in einen wartenden Zustand über.

Empfängt der Zielhost eine Nachricht mit `msg_id=mig_success_notify`, so wird der Zielhost vom Quellhost darüber informiert, dass die tatsächliche Migration erfolgreich beendet wurde. Der Zielhost überprüft die Nachricht wieder auf Richtigkeit, indem er die Liste der aktuellen Migrationsvorgänge auf Existenz der `mig_id` prüft und die `msg_id` auf Plausibilität prüft. Wenn diese Tests in Ordnung sind, so leitet der Quellhost die Nachricht unverändert an die VM weiter. Daraufhin wartet der Zielhost auf eine Antwort von der VM. Diese Antwort leitet der Zielhost dann wiederum unverändert an den Quellhost weiter. Tritt bei der Nachbereitung auf der VM ein Fehler auf, so ist in der Antwort von der VM `err_id!=0` und nachdem der Zielhost die Nachricht nur weiterleitet, kann der Quellhost die Nachricht auswerten und eventuell notwendige Aufräumaktionen einleiten. Nachdem der Zielhost dem Quellhost geantwortet hat, muss er die `mig_id` aus der Liste der aktuellen Migrationsvorgänge löschen, um für neue Migrationsvorgänge mit der gleichen `mig_id` bereit zu sein.

5.5 Programmstruktur in der VM

Die VM implementiert, wie der Zielhost einen konkurrenten Server. Die VM empfängt im regulären Verlauf eines Migrationsvorgangs genau drei Nachrichten.

Die erste Nachricht hat die `msg_id=prepare_vm`. Empfängt die VM diese Nachricht, so speichert die VM die `mig_id` in die Variable `current_mig`. Ist diese Variable beim Empfang der Nachricht ungleich Null, so lehnt die VM diese ab und antwortet mit einer Nachricht mit `err_id=EOUTOFORDER`. Andernfalls bereitet die VM eine Migration ihrer selbst vor. Dabei erstellt die VM einen Tunnelendpunkt, der erlaubt, dass der Zielhost die VM bereits jetzt unter der IP-Adresse des Zielnetzes erreicht. Dann veranlasst die Protokollinstanz auf der VM, dass das Server-Programm Subflows zu allen verbundenen Clients aufbaut. Dabei schickt die Protokollinstanz ein Signal an das Server-Programm und schreibt dann die IP-Adresse, die die VM im Zielnetz besitzt in ein FIFO. Das Server-Programm kann mit der Information aus dem FIFO die Subflows hinzufügen und meldet sich dann bei der Protokollinstanz zurück. In dieser Zeit wartet die Protokollinstanz nur auf die Antwort. Diese enthält Informationen, wie viele Subflows erstellt werden konnten, und auch wie viele offene TCP-Verbindungen das Server-Programm hat. Daraus kann eine Quote berechnet werden, wie viele TCP-Verbindungen nach der Migration noch erhalten bleiben. Nach der erhaltenen Rückmeldung zerstört die VM ihren IP-Tunnelendpunkt wieder und ordnet der Netzschnittstelle, über die die VM erreichbar ist, die IP-Adresse im Zielnetz zu. Dann veranlasst das Protokoll das Zerstören des IP-Tunnelendpunkts auch auf dem Zielhost. Dazu schickt die VM eine Nachricht mit `msg_id=subflow_ready` an den Zielhost. Daraufhin wartet die VM auf eine Antwort, die zweite Nachricht, die die VM während des Ablaufs erhält. Während des Wartens wird die Verbindung zum Zielhost nicht abgebaut, es muss also die `mig_id` der Antwort nicht geprüft werden. Es wird aber die `msg_id` und die `err_id` auf Validität geprüft. Wenn `msg_id!=tunnel_down` oder `err_id!=0`, so schickt die VM diese Nachricht an den Quellhost weiter, um auf den Fehler aufmerksam zu machen. Meldet die Antwort vom Zielhost allerdings Erfolg, so schickt die VM eine Nachricht mit `msg_id=vm_ready` an den Quellhost. Die VM ist an diesem Punkt bereit migriert zu werden. Dann wartet die VM auf weitere eingehende Nachrichten.

Die dritte Nachricht, die die virtuelle Maschine im regulären Protokollablauf erhält besitzt die `msg_id=mig_success_notify`. Diese erhält die VM vom Zielhost. Die VM prüft bei Erhalt dieser Nachricht, ob die `mig_id` der Nachricht mit der intern in `current_mig` gespeicherten Migrations-ID übereinstimmt. Ist das nicht der Fall, so antwortet die VM mit `err_id=EOUTOFORDER`. Andernfalls weiß die VM, dass sie die korrekte Nachricht empfangen hat und somit auch, dass sie erfolgreich migriert wurde und bereits auf dem neuen Host läuft. In dem Fall muss die VM ihr Default-Gateway anpassen, sodass der Netzwerkverkehr standardmäßig über das Bridging-Interface vom Zielhost geroutet wird und nicht mehr über das des Quellhosts. Erst ab dem Moment ist die migrierte VM im Zielnetz ohne IP-Tunnel erreichbar. Nun benachrichtigt die Protokollinstanz auf der VM das Server-Programm mittels Signal, dass die nicht mehr benötigten Subflows, die die VM unter ihrer alten IP-Adresse erreicht haben, abgebaut werden können. Dieser Schritt ist nicht zwingend notwendig, ist aber Teil einer sauberen Implementierung. Die Protokollinstanz erwartet aber keine Rückmeldung vom Server-Programm und schickt sofort nach dem Senden des Signals die Nachricht mit `msg_id=mig_success_ack` an den Zielhost zurück.

Der Protokollablauf für die VM ist an diesem Punkt am Ende. Die VM löscht den Inhalt der `current_mig`-Variable und wartet auf den Start eines neuen Migrationsvorgangs.

6 Evaluierung

Nachdem der Protokollentwurf und die Implementierung des Protokolls zur verbindungserhaltenden Migration virtueller Maschinen im WAN gezeigt wurde, folgt hier die Auswertung der erzielten Ergebnisse. Zuerst wird das Programm, das die Verbindungserhaltung zeigt, erklärt. Anschließend werden erzielte Ergebnisse mit den Anforderungen abgeglichen und auf Erfüllung geprüft.

6.1 TCP-Ping-Pong

Um die verbindungserhaltende Migration zu zeigen, wurde ein Server-Client Programm implementiert, das zeigt, dass die Verbindung des Clients zur VM während der Migration nicht abbricht. Der Teil des Programms, der auf dem Client läuft, wird „Ping-Client“ genannt, der Teil, der auf dem Server läuft „Ping-Server“. Der Ping-Client sendet in festen Zeitintervallen Anfragen an den Server, die dieser beantwortet. Durch das Senden der Antworten des Ping-Servers wird die Erreichbarkeit des Servers gezeigt.

6.1.1 Ping-Client

Der Ping-Client wird gestartet mit der IP-Adresse des Servers als Kommandozeilenparameter. Dann versucht der Client eine TCP-Verbindung zu diesem Server aufzubauen. Gelingt dies, so schickt der Client den String `ping-request` an den Server und wartet auf eine Antwort des Servers. Ist diese Antwort eingetroffen, so wartet der Client eine bestimmte Zeit, bis es wieder an die Stelle springt, an der `ping-request` an den Server geschickt wird. Dies macht der Client solange, bis er das Signal `SIGINT` erhält. Dann schickt der Client die Nachricht `end` an den Server, um dem Server mitzuteilen, dass er die Verbindung beenden will. Anschließend wird der Socket geschlossen und somit die TCP-Verbindung abgebaut.

6.1.2 Ping-Server

Der Ping-Server ist ein konkurrenter Server. Er ist in der Lage, mehreren Clients, die Kontakt zu ihm aufnehmen gleichzeitig zu antworten. Das Programm wird ohne Kommandozeilenparameter gestartet und läuft als Service auf der VM, die verbindungserhaltend migriert werden soll. Das Ping-Server Programm ist die Instanz, die das Hinzufügen und das Abbauen von Subflows im Quellcode implementiert. Nur der Ping-Server weiß, mit wievielen und mit welchen Clients er verbunden ist und ist so der Einzige, der diese Aufgabe übernehmen kann.

Nach dem Start des Programms wartet der Ping-Server auf eingehende Verbindungen. Beim Erhalt einer Verbindungsanfrage wird diese akzeptiert und ein Kindprozess erstellt, der die Clientanfrage bearbeitet. Nun erhält der Server in regelmäßigen Abständen den String `ping-request` und sendet an den Client den String `ping-reply` zurück. Erhält der Server den String `end`, so schließt er den Socket und beendet den Kindprozess.

Der Elternprozess pflegt eine Liste von verbundenen Clients. Sobald eine neue Verbindungsanfrage eingeht, wird der Client in die Liste der verbundenen Clients mit aufgenommen. Sobald ein Kindprozess, der einen Client bedient, terminiert, wird dieser Client aus der Liste der verbundenen Clients gelöscht. Somit kann sichergestellt werden, dass Subflows zu allen verbundenen Clients aufgebaut werden können, sobald der Ping-Server aufgefordert wird, dies zu tun.

Wenn die VM migriert werden soll, und das in Kapitel 4 beschriebene Protokoll auf der VM im Zustand „prepare_vm received“ ist, so schickt das Steuerungsprogramm das Signal `SIGUSR1` an die Ping-Server Applikation. Dann liest dieses im nächsten Schritt die IP-Adresse der VM im Zielnetz aus einer Named Pipe und fügt dann die notwendigen Subflows zu den verbundenen Clients hinzu. Dann schickt die Ping-Server Applikation das Signal `SIGUSR1` zurück an das Steuerungsprogramm. Zusätzlich schreibt der Ping-Server die Zahl der Subflows, die geöffnet werden konnten und die Zahl der verbundenen Clients in eine Named Pipe, sodass das Steuerungsprogramm weiß, wieviele Subflows von wievielen verbundenen Clients aufgebaut werden konnten. Dann wird die VM migriert und das Steuerungsprogramm meldet sich das nächste mal bei dem Ping-Server, wenn die alten Subflows abgebaut werden können. Dazu schickt das Steuerungsprogramm das Signal `SIGUSR2`. Dann baut der Ping-Server die alten Subflows ab. Eine Rückmeldung wird nicht gegeben, da dieser Schritt nicht notwendig ist für den Erfolg der verbindungserhaltenden Migration.

Wenn der Server nach den erläuterten Aktionen und nach der Migration für die vor der Migration verbundenen Clients noch erreichbar ist, so ist die verbindungserhaltende Migration erfolgreich abgelaufen. Erreichbar bedeutet, der Server schickt auf die von den Clients kommenden `ping-request`s nach wie vor `ping-reply`s.

6.2 Auswertung der Ergebnisse

Die in Kapitel 2 erarbeiteten Anforderungen werden in diesem Abschnitt mit den erzielten Ergebnissen verglichen. Tabelle 2.1 gibt einen Überblick über die Anforderungen, die an das Protokoll gestellt wurden.

F1 Der in Abschnitt 6.1 beschriebene Ping-Server und Ping-Client wurde entwickelt, um zu sehen, dass die bestehenden TCP-Verbindungen während einer Migration im WAN nicht abbrechen. Dies geschieht durch eine Anfrage des Ping-Clients, die vom Ping-Server beantwortet wird. Wird diese Anfrage auch nach der Migration vom Ping-Server beantwortet, so konnte die bestehende TCP-Verbindung erhalten werden. Diese Anforderung wird erfüllt, der Ping-Server bleibt für den Ping-Client im ganzen Migrationsverlauf, und insbesondere auch nach der Migration erreichbar. Dieser Testfall wurde auch mit mehreren Ping-Clients, die mit dem Ping-Server verbunden sind durchgeführt. Der Ping-Server ist in der Lage, auch mehrere TCP-Verbindungen zu verschiedenen Clients durch die Migration hindurch zu erhalten.

F2 Die Sitzungsverwaltung fordert vom Quellhost und Zielhost, dass mehrere Protokollabläufe nebenläufig stattfinden können. Darüber hinaus darf auf der VM immer nur eine Instanz des Protokolls zur gleichen Zeit ablaufen dürfen.
Betrachten wir zuerst den Quellhost. Dieser generiert beim Start einer Protokollinstanz eine `mig_id`. Diese stellt sicher, dass unterschiedliche Protokollinstanzen unterschiedliche `mig_id`s besitzen. Somit kann bei einer eingehenden Nachricht anhand dieser ID

abgelesen werden, zu welchem Migrationsvorgang sie gehört und dann vom Quellhost dementsprechend verarbeitet werden. Die Implementierung zeigt, dass die Verwendung der `mig_id` den erwünschten Effekt bringt. Es konnten erfolgreich zwei Ping-Server gleichzeitig migriert werden. Lediglich der Schritt der Migration, bei dem die Pages einer VM auf den Zielhost kopiert werden, kann nicht parallel ablaufen. Dies liegt an der Implementierung des XL-Toolstack. Die Protokollfunktionen werden davon aber nicht beeinflusst, diese laufen parallel ab.
Der Zielhost wertet, wie der Quellhost, die `mig_id` der empfangenen Nachricht aus. Das gibt ihm die Möglichkeit zwischen verschiedenen Protokollinstanzen zu unterscheiden und somit, mehrere Migrationsvorgänge gleichzeitig zu behandeln. Der Zielhost wurde als konkurrenter Server implementiert. Somit kann er mehrere Anfragen gleichzeitig bearbeiten. Bei Erhalt einer Nachricht prüft er die `mig_id` der Nachricht und erkennt daran um welchen Migrationsvorgang es sich handelt. Die gleichzeitige Migration zweier Ping-Server vom Quellhost auf den Zielhost konnte erfolgreich durchgeführt werden. Der Zielhost bearbeitete eingehende Anfragen beider Migrationsvorgänge gleichzeitig.
Die VM verwendet, wie die beiden anderen beteiligten Maschinen, ebenfalls die `mig_id`, um verschiedene Migrationsvorgänge zu unterscheiden. Allerdings akzeptiert die VM keine Anfragen, deren ID nicht der `mig_id` der aktuell ablaufenden Migration ist. Konkret implementiert auch die VM einen konkurrenten Server, der die `mig_id` prüft und auf Anfragen mit falscher ID einen Fehler zurückgibt. Getestet wurde diese Funktion, indem eine VM gleichzeitig von zwei Protokollinstanzen vom Quellhost auf den Zielhost migriert werden sollte. Hierbei muss darauf geachtet werden, dass diese Protokollinstanzen unterschiedliche `mig_id`s verwenden, sonst wird der Vorgang abgebrochen, bevor die VM eine zweite Migrationsanfrage erhält. Dieses Testszenario lieferte das erwartete Ergebnis, die VM lehnt weitere Anfragen, die nicht zur aktuell ablaufenden Migration gehören ab und schickt dem Sender der Anfrage einen Fehler.

F3 Die Fehlererkennung wird von dem Protokoll mithilfe einer Fehler-ID realisiert. Diese ist Teil einer jeden ausgetauschten Nachricht. Tritt ein Fehler im Protokollablauf auf, so wird diese Fehler-ID auf einen für den Fehler vordefinierten Wert gesetzt. So ist es im Protokoll möglich, den Fehler zu identifizieren und sich von dem Fehler zu erholen. Der Prototyp wurde getestet, indem die denkbaren Fehler bewusst hervorgerufen wurden und das Verhalten des Protokolls im Fehlerfall beobachtet wurde. Der Prototyp war in allen evaluierten Fehlerfällen in der Lage, den Ursprungszustand von vor dem Anstoß des Migrationsvorgangs herzustellen. Auch wurde sichergestellt, dass im Falle einer Verbindungsunterbrechung der Kommunikationspartner nicht für unbestimmte Zeit auf eine Antwort der vom Netz abgeschnittenen Komponente wartet. Auch dieser Fehler wurde auf dem Prototypen bewusst hervorgerufen und beobachtet. Hierbei musste zusätzlich überprüft werden, ob die beiden im Netz erreichbaren Maschinen im Ausgangszustand hinterlassen werden. Dies war bei den durchgeführten Tests ebenfalls der Fall.

F4 Das Protokoll verfügt über eine Schnittstelle zur Server-Applikation in der VM. Die Implementierung des Protokolls zeigt die erfolgreiche Umsetzung dieser Schnittstelle. Bei jedem Migrationsvorgang kommuniziert die Protokollinstanz auf der VM mit der Server-Applikation um diese zum Aufbau der benötigten Subflows anzustoßen und ebenso um das Schließen der überflüssigen Subflows nach der geglückten Migration zu

veranlassen. Getestet wurde diese Funktionalität, indem die Kommunikation zwischen den beiden Instanzen während eines Migrationsvorgangs beobachtet wurde.

NF1 Eine sinnvolle Nutzung der Ressourcen wurde sichergestellt, indem beim Entwurf des Protokolls darauf geachtet wurde, sowohl lokale Berechnungen, als auch Nachrichten über das Netzwerk auf ein Minimum zu beschränken. So wird beispielsweise der notwendige IP-Tunnel nach dem Hinzufügen der Subflows wieder abgebaut, um keinen weiteren unnötigen Datenverkehr über den Tunnel zu routen. Der reibungslose Ablauf eines Migrationsprozesses und die Tatsache, dass die Zeit, die ein Migrationsvorgang benötigt, zum größten Teil vom Kopieren der Speicher-Pages bestimmt ist, zeigt, dass das Protokoll unter Berücksichtigung von sinnvoller Ressourcennutzung entwickelt und implementiert wurde.

NF2 Es wurde versucht ein Protokoll zu entwickeln, das von einem sauberen Design charakterisiert ist. Realisiert wird dies beispielsweise durch die oben beschriebene Fehlerbehandlung aber auch durch das Abbauen nicht mehr benötigter Subflows. Der Erfolg eines Migrationsablaufs würde nicht beeinträchtigt werden, wenn die nicht mehr benötigten Subflows nicht abgebaut würden. Trotzdem wird dieser zusätzliche Schritt in Kauf genommen, um die TCP-Verbindungen in einem sauberen Zustand zu hinterlassen.

ID	Anforderung	Status
funktionale Anforderungen		
F1	Erhalt der bestehenden TCP-Verbindungen	✓
F2	Sitzungsverwaltung	✓
F3	Fehlererkennung und -rehabilitation	✓
F4	Schnittstelle zur Server-Applikation	✓
nicht-funktionale Anforderungen		
NF1	Effizienz	✓
NF2	Robustheit	✓

Tabelle 6.1: erfüllte Anforderungen an das Protokoll

Zusammenfassend konnten die an das Protokoll gestellten Anforderungen umgesetzt werden und im Zuge des Prototyps auch implementiert werden. Die Tabelle 6.1 zeigt eine Übersicht der Anforderungen und deren Status. Die verbindungserhaltende Migration von virtuellen Maschinen im WAN ist mit allen gestellten Anforderungen geglückt und realisierbar. Ein bedauerlicher Umstand ist es allerdings, dass neben der Verwendung des Steuerungsprotokolls immer noch die Server-Programme, die über TCP-Verbindungen zu Clients verfügen verändert werden müssen. Diese müssen zum einen die Subflows zu allen Clients hinzufügen, zum anderen mit dem Protokoll kommunizieren. Eine TCP-Verbindung kann nur von dem Programm geändert werden, das die TCP-Verbindung aufgebaut hat, nicht aber von einem anderen Programm. Wenn man allerdings diese Einschränkung in Kauf nimmt, ist dies eine elegante Lösung, eine VM verbindungserhaltend im WAN zu migrieren.

7 Fazit und Ausblick

Das Ziel der Arbeit war es, ein Protokoll zu entwickeln, das eine verbindungserhaltende Migration virtueller Maschinen im WAN ermöglicht. Dazu wurden in Kapitel 2 Anforderungen an das Protokoll gestellt, die eine spätere Nützlichkeit des Protokolls sicherstellen. In Kapitel 3 wurde ein Blick auf den aktuellen Stand der Technik geworfen, um festzustellen, dass schon Forschungen in diese Richtung existieren, allerdings keiner dieser Lösungsansätze den gestellten Anforderungen genügt. Im darauf folgenden Kapitel 4 wurde ein Protokoll entwickelt und vorgestellt, das die definierten Anforderungen erfüllt und somit die Lücke an Funktionalität schließt. Dieses Protokoll wurde dann in Kapitel 5 implementiert. Kapitel 6 zeigt zusätzlich die Implementierung eines Server-Client Programms, das den Erhalt der Verbindung während der Migration einer VM zeigt. Außerdem werden die erzielten Ergebnisse in diesem Kapitel aufgeführt und rückblickend mit den Anforderungen abgeglichen.

Zusammenfassend kann man feststellen, dass das Protokoll eine Verbesserung des Status Quo darstellt. Es steuert die verbindungserhaltende Migration von virtuellen Maschinen zwischen verschiedenen Broadcast-Domänen. Um sicherzustellen, dass die bestehenden TCP-Verbindungen zu den Clients nicht unterbrochen werden, nutzt das Protokoll MPTCP. Damit werden jeder TCP-Verbindung neue Subflows hinzugefügt, die es ermöglichen, dass der Client die VM unter der IP-Adresse der VM im Zielnetz erreicht. Der Subflow von Client zur VM im Ursprungsnetz wird nach der Migration abgebaut. Indem durch MPTCP neue Routen als Subflows hinzugefügt werden, und der Netzwerkverkehr nicht vom Ursprungsnetz in das Zielnetz getunnelt wird, wird eine minimale Netzauslastung erreicht.

Mit der Nutzung dieses Protokolls ist es möglich, die Live-Migration virtueller Maschinen nicht nur im LAN zu nutzen, sondern auf das WAN zu erweitern. Dadurch, dass die TCP-Verbindungen von VM zu den Clients nicht abbrechen, ist diese Migration für die Clients transparent. Das ermöglicht es Anbietern von Cloud-Diensten die VMs der Kunden flexibel auf Hosts in Rechnernetzen verteilt über die ganze Welt zu migrieren, ohne dass der Kunde davon etwas mitbekommt. Zusätzlich wurde auf die Skalierbarkeit des Systems geachtet. So ist es möglich, eine Vielzahl von VMs von einem Quellhost auf einen oder mehrere Zielhosts zu migrieren. Damit muss der Systemadministrator die Migration von mehreren VMs nicht für jede VM einzeln anstoßen. Dadurch wird sichergestellt, dass das Protokoll auch in Produktivumgebungen mit mehreren hundert VMs auf einem physischem Host von Nutzen ist.

Trotzdem bietet das Protokoll auch einige Ansatzpunkte für Verbesserungen, beziehungsweise weiterführende Arbeiten, die im Rahmen dieser Arbeit nicht abgedeckt wurden.

Algorithmus zur Entscheidung über das Stattfinden der Migration: Das Protokoll bekommt auf VM-Seite von dem Server-Programm eine Rückmeldung, wieviele von den bestehenden TCP-Verbindungen mit einem neuen Subflow versehen werden konnten. Diese Information wird aktuell vom Protokoll nicht verwertet. Es wäre denkbar, diese Information mit dem Quellhost zu teilen, um dann anhand der Quote der erfolgreich hinzugefügten Subflows zu entscheiden, ob die Migration stattfinden soll oder nicht.

Diese Entscheidung muss nicht zwangsläufig mit einer Nutzerinteraktion stattfinden. Auch kann ein Algorithmus entwickelt werden, der komplett selbstständig oder anhand von initial vorhandenen Nutzeranforderungen entscheidet, ob die Migration stattfindet oder nicht.

Verbesserte Fehlerbehandlung: Eine weitere Verbesserung des Protokolls wäre eine Verfeinerung der Fehlerbehandlung. Diese wurde derartig entworfen, dass auftretende Fehler erkannt und gemeldet werden, die Migration aber dabei abgebrochen wird und der Ausgangszustand wieder hergestellt wird. Diese Fehlerbehandlung könnte dahingehend optimiert werden, indem der Migrationsvorgang beim Auftreten eines Fehlers nicht abgebrochen wird. Dann muss versucht werden, den auftretenden Fehler zu eliminieren. Dies kann durch einfaches wiederholtes Durchführen der Aktion, die den Fehler verursacht hat, erreicht werden oder durch das Finden und Beheben des Umstandes, der den Fehler verursacht hat.

Programmierschnittstelle (API): Darüber hinaus ist es denkbar, eine API als Schnittstelle zwischen Protokoll und Server-Anwendung zu entwickeln. Die in dieser Arbeit gezeigte Implementierung realisiert eine sehr grundlegende Kommunikation zwischen der Protokollinstanz auf der VM und der Server-Anwendung. Die Server-Anwendung muss in der Lage sein, Nachrichten von der Protokollinstanz zu empfangen, umzusetzen und eine valide Antwort zu schicken. Außerdem muss die Server-Applikation die neuen Subflows öffnen und auch wieder schließen können. Um dies zu erreichen, muss die Server-Anwendung manuell verändert werden, um mit dem Protokoll arbeiten zu können. Wenn Anwendungsentwicklern eine Schnittstelle zur Kommunikation mit dem Protokoll vorliegt, so können diese ihre Anwendung mit Unterstützung dieser API entwickeln und die Anwendung muss nicht mehr manuell bearbeitet werden, um das Protokoll zu unterstützen.

Hochskalierter Anwendungsfall: Als weiterführende Arbeit ist es denkbar, den Anwendungsfall der verbindungserhaltenden Migration im WAN hochzuskalieren und unter Last zu evaluieren. Ein denkbarer Anwendungsfall wäre die Migration eines kompletten Rechenzentrums. Hierbei müssen Effekte berücksichtigt werden, die bei der Migration einer oder weniger virtueller Maschinen keine Rolle spielen. So muss beispielsweise die Netzauslastung überwacht werden um sicherzustellen, dass die Netzinfrastruktur nicht lahmgelegt wird.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[CFH+05] Clark, Christopher ; Fraser, Keir ; Hand, Steven ; Hansen, Jacob G. ; Jul, Eric ; Limpach, Christian ; Pratt, Ian ; Warfield, Andrew: Live Migration of Virtual Machines. In: *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*. Berkeley, CA, USA : USENIX Association, 2005 (NSDI'05), 273–286

[deb18] *Debian – The Universal Operating System*. `https://www.debian.org`. Version: May 2018. – [Online; accessed 28. May 2018]

[HB16] Hesmans, Benjamin ; Bonaventure, Olivier: An enhanced socket API for Multipath TCP. In: *Proceeding ANRW '16 Proceedings of the 2016 Applied Network*, 2016

[mpt18a] *MultiPath TCP - Developper Mailing List*. `https://listes-2.sipr.ucl.ac.be/sympa/info/mptcp-dev`. Version: Apr 2018. – [Online; accessed 20. Apr. 2018]

[mpt18b] *MultiPath TCP - Linux Kernel implementation*. `https://www.multipath-tcp.org`. Version: Apr 2018. – [Online; accessed 20. Apr. 2018]

[nbd18] *NetworkBlockDevice*. `https://github.com/NetworkBlockDevice`. Version: May 2018. – [Online; accessed 9. May 2018]

[PB14] Paasch, Christoph ; Bonaventure, Olivier: Multipath TCP. In: *Queue* 12 (2014), Februar, Nr. 2, 40:40–40:51. `http://dx.doi.org/10.1145/2578508.2591369`. – DOI 10.1145/2578508.2591369. – ISSN 1542–7730

[Rod08] Rodenhäuser, Ben: Cloud Computing: Damit Sie nicht aus allen Wolken fallen. In: *manager magazin* (2008), Oct. `http://www.manager-magazin.de/unternehmen/it/a-582750-2.html`

[TPDG+06] Travostino, F. ; P-Daspit ; Gommans, L. ; Jog, C. ; Laat, C. de ; Mambretti, J. ; Monga, I. ; Oudenaarde, B. van ; Raghunath, S. ; Wang, P.: Seamless live migration of virtual machines over the MAN/WAN. In: *Future Generation Computer Systems* 22 (2006), S. 901–907

[Vera] Verhelst, Wouter: *NBD-CLIENT(8) Manual Page*

[Verb] Verhelst, Wouter: *NBD-SERVER(1) Manual Page*

[xen18] *Xen Wiki*. `https://wiki.xenproject.org/wiki/Main_Page`. Version: Jan 2018. – [Online; accessed 19. Apr. 2018]

[xl17] *XL - Xen*. `https://wiki.xenproject.org/wiki/XL`. Version: Apr 2017. – [Online; accessed 19. Apr. 2018]