IPv6 in OMNeT++

B. Sc. Thomas Karl

Hochschule Bonn-Rhein-Sieg, Fachbereich Informatik,
Wintersemester 2011/2012

thomas.karl@smail.inf.h-brs.de

Abstract

Das Simulationswerkzeug OMNeT++ eignet sich durch die Zuhilfenahme des INET-Frameworks zur Simulation von Kommunikationsnetzen. Im Rahmen dieser Arbeit erfolgt eine Bestandsanalyse der IPv6-Funktionalität im INET-Framework. Eine kurze Einführung in OMNeT++ und INET soll dem Verständnis der wesentlichen Bestandteile dienen. Durch die Entwicklung eines einfachen Testszenarios soll der bisherigen Entwicklungsstand verifiziert sowie eine Aussage über die Einsatzmöglichkeiten von OMNeT++ und INET im Umfeld von IPv6 getroffen werden.

Keywords

OMNeT++, INET-Framework, IPv6, Neighbor Discovery Protocol (NDP), IPv6 Stateless Address Autoconfiguration (SAC)

1. Einführung in OMNet++

OMNeT++ [1] ist ein diskretes, eventbasiertes C++ Framework zur Modellierung von Netzsimulationen. OMNeT++ ist für den akademischen und nichtkommerziellen Nutzen frei verfügbar und unter der „Academic Public License“ lizensiert. Statt der direkten Bereitstellung von Simulationskomponenten, beispielsweise für Kommunikationsnetze oder Warteschlangensysteme, bietet OMNeT++ eine grundlegende Maschinerie und Werkzeuge zur Entwicklung von Simulationen. Dazu gehört beispielsweise eine auf Eclipse basierende IDE, eine grafische Benutzeroberfläche zur Ausführung von Simulationen oder aber eine eigene Beschreibungssprache zur Modellierung einer Simulation (NED). Die Komponenten für spezifische Applikationsbereiche werden durch Frameworks bzw. Simulationsmodelle bereitgestellt. [2]

In den folgenden drei Abschnitten werden die wesentlichen Bestandteile von OMNeT++ erläutert, die für das weitere Verständnis dieser Arbeit von Bedeutung sind. Dabei wird vor allem auf die fachspezifische Terminologie von OMNeT++ eingegangen.

1.1. Modellstruktur

In OMNeT++ besteht ein Simulationsmodell aus einer Menge von Modulen die untereinander verbunden sind (*connections*) und über Nachrichten (*messages*) miteinander kommunizieren. Aktive Module eines Simulationsmodells werden als *simple module* bezeichnet. Sie versenden, empfangen und verarbeiten Nachrichten an definierten Eingangs- und Ausgangsschnittstellen (*gates*). Einfache Module sind in C++ geschrieben und greifen zur Funktionsweise auf eine C++ Klassenbibliothek von OMNeT++ zurück. Mehrere einfache Module (*simple modules*) können zu einem zusammengesetzten Modul (*compund module*) gruppiert werden. Dadurch ist eine hierarchische Definition eines Simulationsmodells möglich, wodurch auch komplexe Strukturen übersichtlich darstellbar sind. Das komplette Simulationsmodell - in OMNeT++ als *Network* bezeichnet - ist letztlich nichts anderes als ein zusammengesetztes Modul. Die Zusammenhänge der einzelnen Bestandteile sind nochmals in Abbildung 1 dargestellt (einfache Module besitzen einen grauen Hintergrund). [3]

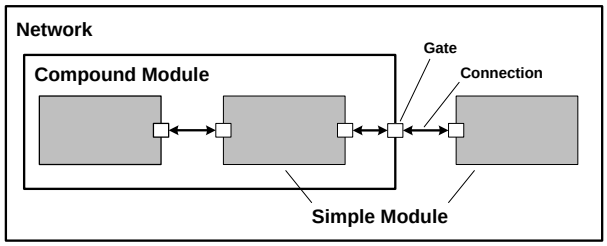

Abbildung 1: Modellstruktur in OMNeT++

Zur Definition der Struktur eines Modells, wie sie in Abbildung 1 gezeigt wurde, steht dem Anwender in OMNeT++ eine eigene Beschreibungssprache zur Verfügung. Diese wird als *NED language* bezeichnet (Network Description language).

1.2. Network Description Language (NED)

OMNeT++ nutzt zur Definition der Topologie eines Simulationsmodells eine eigene Beschreibungssprache. Diese wird als NED (Network Description) bezeichnet. Typischerweise wird die NED-Beschreibungssprache genutzt um einfache Module zu deklarieren und zusammengesetzte Module

sowie komplette Netzwerke zu definieren. Die Deklaration eines einfachen Moduls dient der Beschreibung seiner Schnittstellen: *Gates* und *Parameters*. Bei der Definition von zusammengesetzten Modulen werden ebenfalls die externen Schnittstellen (*Gates* und *Parameters*) beschrieben sowie Submodule und ihrer Verbindungen (*connections*) untereinander bestimmt. Bei den Netzwerken können zusätzlich noch Übertragungskanäle definiert werden, wodurch eine Abbildung von verschiedensten Übertragungsmedien mit ihren charakteristischen Eigenschaften (Delay, Jitter, Packetloss, etc.) möglich wird. [2]

OMNeT++ bietet zwei Möglichkeiten zur Arbeit mit der NED-Beschreibungssprache. Zum einen lassen sich NED-Dateien mit den zuvor beschriebenen Inhalten über den in der IDE integrierten grafischen NED-Editor definieren, zum anderen ist es aber auch möglich die einzelnen Teile direkt in textueller Form über einen beliebigen Texteditor zu erstellen.

1.3. Architektur eines Simulationsprogramms

Die Architektur einer Simulation ist modular gestaltet und in der nachfolgenden Abbildung dargestellt:

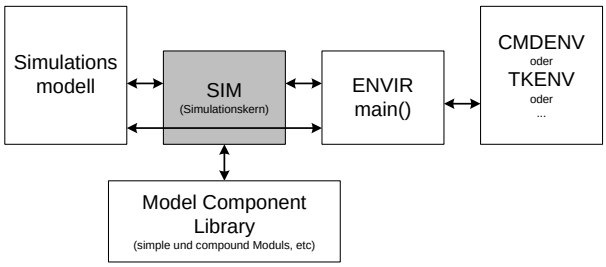

Abbildung 2: Architektur einer OMNeT++ Simulation

Das Zentrum einer solchen Simulation bildet der in grau dargestellte Simulationskern (*SIM*). Die „*Model Component Library“* beinhaltet die einfachen Module mit ihren zugehörigen C++ Implementierungen (kompiliert), zusammengesetzte Module, Kanäle, Netzwerke, Nachrichtentypen und alles weitere was zu Modellen gehört, die mit der Simulation verlinkt sind. Vor Beginn der Simulation erzeugt der Simulationskern *SIM* das konkrete *Simulationsmodell*. Hier werden alle für den Simulationsablauf benötigten Objekte (Module, Kanäle, etc.) aus der „*Model Component Library“* instanziiert. Welche Objekte dies sind, wird von *ENVIR* durch die Interpretation der NED-Files vorgegeben. Die eigentliche Ausführung der Simulation erfolgt über die *main()*-Funktion in der Laufzeitumgebung *ENVIR*. Die Steuerung von einzelnen Parametern und das Verhalten von Objekten einer Simulation erfolgt über eine INI-Datei (typischerweise omnetpp.ini). Darüber lässt sich dann beispielsweise der Start einer bestimmten Applikation definieren. Der Nutzer selbst hat die Möglichkeit die Simulation über verschiedenste Benutzerschnittstellen zu analysieren oder zu beeinflussen. *CMDENV* oder *TKENV* sind zwei exemplarische Implementierungen einer Nutzerschnittstelle. [2] [3]

2. Das INET-Framework

Da OMNeT++ selbst lediglich die grundlegende Maschinerie und Werkzeuge zur Modellierung von Simulationen bietet, ist zur Erstellung von Kommunikationsnetzen eine Erweiterung notwendig. Die Möglichkeit zur Simulation von Kommunikationsnetzen wird bei OMNeT++ über das INET-Framework [4] geboten. INET stellt eine Sammlung von verschiedensten Protokollimplementierungen für Kommunikationsnetze bereit. Dazu gehört unter anderem TCP, UDP, IP, IPv6, PPP oder Ethernet. Diese Protokolle (oder Teilaspekte davon) sind alle als einfache Module (*simple modules*) in OMNeT++ realisiert. Die externen Schnittstellen der Module (*gates* und *parameters*) sind in NED-Dateien beschrieben, die eigentliche Implementierung erfolgt in identisch benannten C++ Klassen. Die einfachen Module können dann über die NED-Beschreibungssprache zu beliebigen zusammengesetzten Modulen kombiniert werden, um so Knoten in Kommunikationsnetzen zu modellieren. Das INET-Framework bietet bereits typische Netzknoten, wie z.B. Switche, Router, Access Points oder Hosts, auf die zurückgegriffen werden kann. Die Integration in die OMNeT++ Entwicklungsumgebung gestaltet sich als sehr einfach. Das INET-Framework muss einmalig kompiliert werden und steht dem Simulationskern dann als Teil der *Model Component Library* (siehe Abbildung 2) bereit. In neuen Simulationen kann durch eine Referenz auf das INET-Framework die volle Funktionalität genutzt werden. [5] [6]

3. IPv6 Funktionalität in INET

Die IPv6 Funktionalität ist in INET durch mehrere zusammenwirkende Module implementiert und ist als Teilprojekt des gesamten Frameworks zu sehen. Die Umsetzung von IPv6 in OMNeT++ war zu Beginn ein von INET unabhängiges Projekt namens IPv6Suite [7]. Durch András Varga (OpenSim Ltd.) und Wei Yang Ng (Monash University) begann im Jahr 2005 die Integration in das INET-Framework. In den folgenden Jahren wurden in mehreren INET-Releases neue Entwicklungsstände zu IPv6 veröffentlicht. Im Juni 2008 erschien das letzte offizielle Update. [8]

Die Funktionalität von IPv6 unterteilt sich in der aktuellen Version des INET-Frameworks in 5 einfache Module (*simple modules*). Vier dieser Module sind im zusammengesetzten Modul (*compound modul*) *NetworkLayer6* gruppiert. Dieses Modul repräsentiert die IPv6-Umsetzung der Vermittlungsschicht des OSI-Referenzmodells. In Abbildung 1 ist das zusammengesetzte Modul *NetworkLayer6*, bestehend aus den vier einfachen Modulen ipv6, *icmpv6*, *ipv6ErrorHandling* und *neighbourDiscovery* dargestellt. [6]

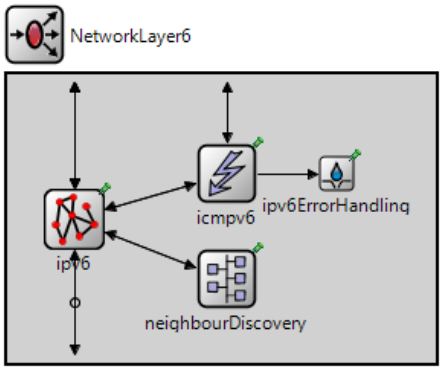

Abbildung 3: Aufbau von NetworkLayer6

Die schwarzen Pfeile repräsentieren dabei Verbindungen (*connections*) zwischen den Modulen. Die Pfeile ausgehend vom Modul *ipv6* zum Rand des grau dargestellten Rechtecks dienen als Schnittstelle zur Transport- bzw. zur Sicherungsschicht. Über die Verbindung vom Modul icmpv6 zum Rand des Rechtecks ist eine Ping-Applikation (PingApp) angebunden. Diese generiert Ping-Anfragen und berechnet aus den Antworten beispielsweise den Paketverlust oder die Round Trip Time (RTT). [6]

Das fünfte und letzte einfache Modul im IPv6-Umfeld von INET heißt *RoutingTable6*. Es besitzt keine Verbindungen zu anderen Modulen und dient lediglich als Speicherort für bestimmte Datenstrukturen.

In den nachfolgenden Abschnitten werden die Aufgaben der fünf einfachen Module kurz erläutert. Es sei hier erwähnt, dass zum überwiegenden Teil der hier vorgestellten Inhalte keine ausführliche Dokumentation existiert. Um Abläufe nachzuvollziehen und Informationen zu bestimmten Modulen zu erlangen, hilft daher oft nur der Blick in den INET-Quellcode und die darin enthaltenen Kommentare.

3.1. ipv6

Dieses Modul beinhaltet den Großteil der Implementierung von IPv6 und dient der Behandlung von IPv6-Pakaten. Dazu gehört beispielsweise das Versenden und Empfangen sowie die Weiterleitung von Paketen an benachbarte OSI-Schichten (Transport- bzw. Sicherungsschicht). Falls dabei ein Datagramm geroutet werden muss, nutzt *ipv6* das einfache Modul *routingTable6* um Informationen über das Ausgangsinterface und die NextHop-Adresse zu erhalten. Dies geschieht nicht über eine Verbindung (connection) zwischen den beiden Modulen, sondern durch den direkten Aufruf von C++-Methoden [6]. Die Modellierung von IPv6-Paketen ist im INET-Framework allerdings unvollständig. Einige Erweiterungsheader (Extension Header) sind im aktuellen Release nicht implementiert. Dazu gehören der *AuthenticationHeader*, der *EncapsulationSecurityPayloadHeader* oder der *DestinationOptionsHeader*. Daher ist eine Umsetzung von Szenarien mit diesen Optionen von IPv6 nicht möglich.

3.2. icmpv6

Das Modul icmpv6 repräsentiert die Implementierung des gleichnamigen Protokolls. Es besitzt dabei eine Verbindung zu einer Ping-Applikation (PingApp) von der es Anfragen zur Generierung von ICMP-Requests entgegennimmt und die Ergebnisse wiederum an diese übergibt. [6]

3.3. neighbourDiscovery

In diesem Modul sind alle Dinge implementiert, die mit der Nachbarschaftsermittlung (Neighbour Discovery Protocol - NDP) und der Stateless Address Autoconfiguration (SAC) verbunden sind. Die Neighbour Discovery-Datenstruktur ist dabei in das Modul RoutingTable6 ausgelagert. Dazu gehören der Destination-Cache, der Neighbour-Cache sowie die Präfix-Liste. Neighbour Discovery-Pakete werden von diesem Modul versendet und empfangen, d.h. das einfache Modul *ipv6* leitet diese Nachricht über eine Verbindung (*connection*) direkt an *neighbourDiscovery* weiter. [6]

3.4. ipv6ErrorHandling

Dieses Modul wird genutzt um Fehlermeldungen für IPv6 auszugeben. Die Fehlermeldungen werden momentan nicht weiter interpretiert. Das Modul ist als Übergangslösung gedacht. Zukünftig soll es eine gemeinsame Fehlerbehandlung für IPv4 und IPv6 geben. [6]

3.5. RoutingTable6

Dieses Modul enthält die IPv6 Routing-Tabelle und die Neighbour Discovery-Datenstruktur. Das Modul besitzt dabei keinerlei Schnittstellen (*gates*) zu anderen einfachen Modulen. Über C++-Methoden

können weitere Module beispielsweise Einträge der Routing-Tabelle lesen oder IPv6-Interfaces konfigurieren. Bei der Ausführung einer Simulation benötigt dieses Modul eine XML-Datei mit optional enthaltenen Routing- und Interface-Informationen. [6]

Abschließend soll zur IPv6 Funktionalität im INET-Framework noch kurz das Projekt xMIPv6 [9] erwähnt werden. Dieses Projekt ist eine Erweiterung zum INET-Framework und dient der Implementierung von Mobile IPv6 nach RFC 3775. Es nutzt dazu die vorhandenen fünf einfachen Module und erweitert diese an entsprechenden Stellen. Die eigentliche Funktion der Module bleibt dabei unangetastet, wodurch sich diese Erweiterung völlig transparent verhält, falls die Mobile IPv6-Funktionalität nicht genutzt wird.

4. Konzeption eines IPv6 Testszenarios

Durch die Entwicklung eines Testszenarios soll die Funktionsweise der zuvor beschriebenen IPv6-Bausteine aus dem INET-Framework verifiziert werden. Das nachfolgende minimalistische Szenario eignet sich zur Untersuchung von elementaren Abläufen des Internet Protokolls in der Version 6 (Abbildung 4): Host1 sendet in regelmäßigen Abständen Pings (ICMPv6 Echo-Request) an Host2, der diese beantwortet (ICMPv6 Echo-Reply).

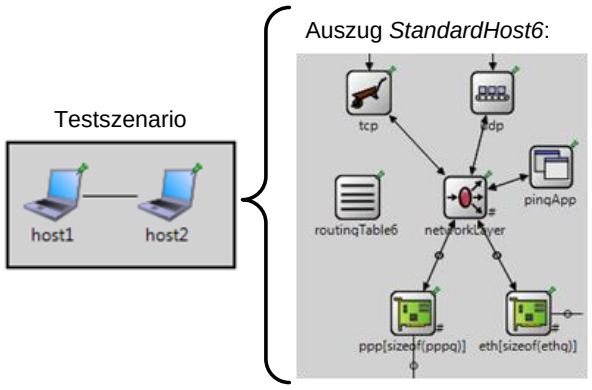

Abbildung 4: IPv6-Testszenario

Die beiden Hosts (*host1* und *host2*) sind dabei zwei Instanzen des zusammengesetzten Moduls *StandardHost6* aus dem INET-Framework. Wie in Abbildung 4 ebenfalls zu sehen, implementiert der StandardHost6 alle der zuvor erläuterten IPv6-Module (*routingTable6* und *NetworkLayer6*). Tendenziell werden bei einem Ping zwischen den beiden Hosts auch alle diese Module benötigt, wie die nachfolgende Erläuterung des groben Ablaufs zeigt:

Zu Beginn erfolgt eine Initialisierung der IPv6-Vermittlungsschicht. Dabei stehen die Generierung von IPv6-Adressen und die Assoziierung zu den entsprechenden Interfaces sowie die Erzeugung von Datenstrukturen (Routing-Tabelle, Interface-Tabelle, etc) im Vordergrund. Für diese Prozedur werden die IPv6-Module *neighbourDiscovery* (u. A. für die Nachbarschaftsermittlung (NDP) und die Stateless Address Autoconfiguration - SAC), *ipv6* (für das Paket-Handling beim SAC) und RoutingTable6 (für die Routing-Tabelle und weitere Datenstrukturen) benötigt. Danach erfolgt das Versenden des ersten Pings ausgehend von Host1 zu Host2. Dazu muss bei Host1 der Ping (ICMPv6 Echo-Request) generiert, der Zielrechner ermittelt und das Paket versendet werden. Host2 muss das Paket entgegennehmen, auswerten und Host1 mit einem ICMPv6 Echo-Reply antworten. Dazu werden in OMNeT++ die Module *pingApp* (zur Generierung und Terminierung des Pings), *icmpv6* (zur Erzeugung der ICMPv6 Echo-Request und Echo-Reply Nachrichten), *ipv6*, *RoutingTable6* und *neighbourDiscovery* (zur Ermittlung des Zielrechners und dem Versenden/Empfangen von NDP-Paketen) benötigt. Das Modul *ipv6ErrorHandling* sollte dabei im Normalfall nicht benötigt werden. Die Funktionsweise ließe sich aber dennoch prüfen, indem beispielsweise ein Ping an eine nicht existente Zieladresse versendet wird. Darauf sollte die ICMPv6-Nachricht *Destination Unreachable* generiert und an das Modul *ipv6ErrorHandling* weitergeleitet werden.

Sollte diese Prozedur ohne Probleme durchlaufen werden, ist das Paket-Handling (Senden, Empfangen, Weiterleiten) als eine elementare IPv6-Funktion gegeben. Ausgehend von diesem minimalistischen Szenario, bietet sich in zukünftigen Untersuchungen eine sukzessive Erweiterung an, bei der weitere IPv6-Funktionen untersucht werden könnten. Durch die Hinzunahme eines Routers und die Aufteilung der beiden Hosts in verschiedene Subnetze könnte beispielsweise das IPv6-Routing betrachtet werden (Abbildung 5). Auch eine Erweiterung des Szenarios um verschiedene Layer 4 Protokolle (TCP, UDP oder SCTP) wäre denkbar.

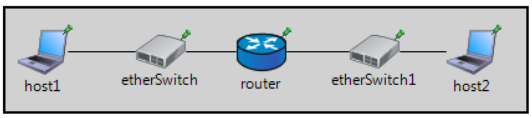

Abbildung 5: Erweitertes IPv6-Testszenario

5. Umsetzung und Analyse

Zur Umsetzung des entwickelten Szenarios ist die Inbetriebnahme der Ping-Applikation (*PingApp*) notwendig. Diese ist bei allen Hosts standardmäßig als Senke aktiviert. Das heißt, dass ICMP-Anfragen jederzeit entgegengenommen und beantwortet werden können. Damit ein Host auch ICMP-Anfragen generiert, muss in der Steuerungsdatei *omnetpp.ini* der Startzeitpunkt der Applikation und die Ziel-IP-Adresse definiert werden [6]. Die Angabe der Ziel-IP-Adresse gestaltet sich dabei als problematisch. Bei der Umsetzung des Internet Protokolls in der Version 4 bietet das INET-Framework im einfachen Modul *RoutingTable* die Möglichkeit statische IPv4-Adressen pro Interface über eine externe Konfigurationsdatei zu definieren [10]. Eine solche Konfigurationsmöglichkeit ist zwar auch im Modul *RoutingTable6* vorgesehen, allerdings ist das Parsen und die Interpretation der Konfigurationsdatei noch nicht umgesetzt. Im Vorgängerprojekt IPv6Suite war diese Möglichkeit geboten, eine Darstellung von IPv6-Parametern ist in [11] zu finden. Die zuständige C++-Methode im INET-Framework zum Parsen der Konfigurationsdatei lautet *configureInterfaceFromXML* und findet sich in der C++-Klasse *RoutingTable6* des INET-Frameworks. Eine alternative Möglichkeit zur Konfiguration der Netzschnittstellen besteht in IPv6 durch die „Stateless Address Autoconfiguration. Dieser Prozess zur automatischen Konfiguration der Schnittstellen ist bei INET im einfachen Modul *neighbourDiscovery* implementiert. Im groben werden bei der Stateless Address Autoconfiguration die folgenden Schritte durchlaufen: [12]

1. **Generierung einer link-lokalen Adresse:**
   Kombination eines Link-Identifiers (z.B.: MAC-Adresse) mit dem link-lokal Präfix FE80::/64.
2. **Überprüfung der Eindeutigkeit:**
   Über Neighbor Solicitation (NS) Nachrichten erfolgt Überprüfung der Adresseindeutigkeit im eigenen Subnetz.
3. **Zuweisung der Adresse zum Interface:**
   Ist die link-lokale Adresse eindeutig (es wird keine Neighbor Advertisement (NA) Nachricht empfangen), wird sie dem Interface zugewiesen. Der Host kann jetzt mit seinen Nachbarn im lokalen Subnetz kommunizieren.
4. **Ermittlung von Routern und Präfix-Abfrage:**
   Der Host ermittelt über die Router Solicitation (RS) Nachricht, welche Router im Subnetz vorhanden sind und welche Präfixe gelten. Ein vorhandener Router antwortet mit einer Router Advertisement (RA) Nachricht.
5. **Generierung einer globalen Adresse:**
   Anhand des übermittelten Präfixes aus vorherigem Schritt (RA-Nachricht), kann sich der Host eine globale Adresse erzeugen.
6. **Zusätzliche Konfigurationsparameter:**
   Der Host erhält vom Router Informationen über weitere Konfigurationsparameter (Standardgateway, Nutzung von DHCPv6, etc.).

Über die ersten drei Schritte besteht damit die Möglichkeit, dass sich die Hosts aus der MAC-Adresse über das EUI-64 Format eine gültige link-lokale IPv6-Adresse generieren. Diese könnte dann für die *PingApp* genutzt werden. Die übrigen Schritte 4 bis 6 der Stateless Address Autoconfiguration sind im aktuellen INET-Release nur teilweise implementiert. Zwar ist die Ermittlung von Routern über RS- und RA-Nachrichten noch möglich, diese können den Hosts aber keine eigens gewählten Präfixe bekanntgeben, da Präfixe ebenfalls über die zuvor erwähnte XML-Konfigurationsdatei definiert werden. Das Host-Konfigurationsprotokoll DHCPv6 ist im aktuellen Release ebenfalls nicht implementiert. Dies bedeutet eine enorme Funktionseinschränkung für IPv6. Es ist mit dem jetzigen Implementierungsstand lediglich eine Kommunikation im eigenen, lokalen Subnetz möglich, da weder eine statische Interface-Konfiguration, noch die Generierung einer globalen IPv6-Adresse über die Stateless Address Autoconfiguration möglich sind.

Mit den bislang ermittelten Informationen ist es nun möglich die *PingApp* zu starten, da als Zieladresse die link-lokale Adresse des benachbarten Hosts verwendet werden kann (hier: fe80::aaa:ff:fe00:2). Bei der Ausführung der Simulation verlassen jedoch keine ICMP-Requests den ersten Host. Zur Problemlösung wird im weiteren Verlauf der eigentliche Ablauf beim Versenden eines IPv6-Pakets beschrieben. Dieser beginnt mit dem Eintreffen eines IPv6-Pakets im Sendepuffer. Die Zieladresse wird dabei mit dem Link-Präfix, welcher sich im Präfix-Cache befindet, abgeglichen. Dadurch kann ermittelt werden, ob sich der Zielrechner im gleichen Subnetz befindet. Falls dies der Fall ist, erfolgt im Neighbor-Cache die Suche nach einem Eintrag mit der Link-Adresse (z.B. MAC-Adresse) des Zielrechners. Existiert dieser Eintrag, wird die Link-Adresse entnommen und zusammen mit dem IPv6-Paket an die Sicherungsschicht übergeben. Andernfalls erfolgt die Ermittlung der Adresse über die ICMPv6-Nachrichten Neighbor Solicitation und Neighbor Advertisement (= NDP - Neighbor Discovery Protocol). Die gewonnene Information wird im Neighbor-Cache hinterlegt. Falls sich der Zielrech-

ner in einem anderen Subnetz befindet, wird über den Destination-Cache die IPv6-Adresse des Routers, der für diese Zieladresse zuständig ist, ermittelt. Aus dem Router-Cache wird dann seine Link-Adresse entnommen. Das IPv6-Paket und die Link-Adresse des Routers werden an die Sicherungsschicht übergeben. In Abbildung 6 ist diese Prozedur in einem Ablaufdiagramm dargestellt.

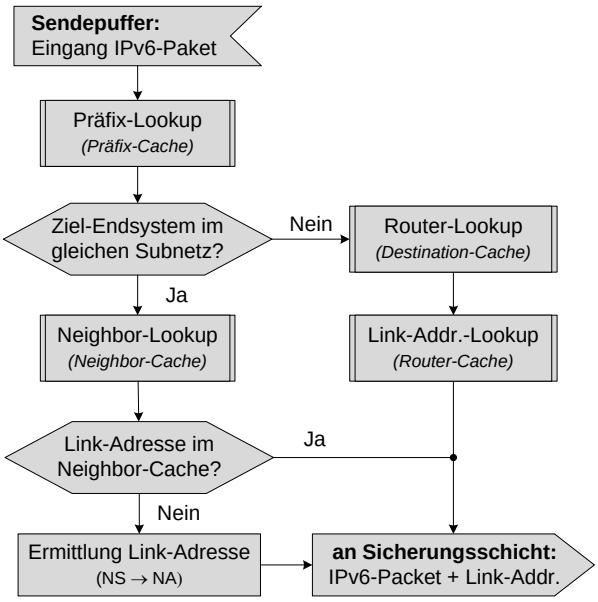

Abbildung 6: Sendevorgang IPv6-Paket (in Anlehnung an Abbildung 7.1-5 in [13])

OMNeT++ bietet mit der Benutzerschnittstelle *TKENV* eine grafische Möglichkeit, Abläufe einer Simulation detailliert zu analysieren. Dabei ist es möglich selbst kleinste Aktionen Schritt für Schritt zu durchlaufen. Durch Breakpoints können zu beliebigen Zeitpunkten die Inhalte jeglicher Datenstrukturen (Neighbor-Cache, Interface-Tabelle, Routing-Tabelle etc.) eingesehen werden. Ein Abgleich mit dem zuvor beschriebenen Ablauf zeigt, dass bereits Probleme beim Präfix-Lookup im aktuellen INET-Release entstehen. Die Zieladresse der ICMP-Nachricht wird vom Modul *ipv6* immer als unbekannte externe Adresse erkannt. Dies liegt daran, dass die Routing-Tabelle des Hosts keine Einträge enthält (Datenstruktur *routeList* im Modul *RoutingTable6*). Bei der Generierung einer link-lokalen Adresse ist die Anpassung der eigenen Routing-Tabelle noch nicht implementiert (Methode *assignLinkLocalAddress* im Modul *neighbourDiscovery*). Eine manuelle Erweiterung der Routing-Tabelle wäre über die bereits mehrfach erwähnte XML-Konfigurationsdatei möglich, welche allerdings in der aktuellen Version nicht interpretiert/ausgewertet wird. Damit gelangt das Paket zum Modul *neighbourDiscovery*. Hier wird die Methode *processIPv6Datagram* durchlaufen. Es wird versucht einen zuständigen Router für diese Zieladresse zu identifizieren, was allerdings erwartungsgemäß fehlschlägt, da die Adresse nur eine lokale Gültigkeit besitzt und daher nicht geroutet werden muss. Dadurch kann kein Ausgangsinterface für das IPv6-Paket ermittelt werden. Im Destination-Cache (Datenstruktur *destCache* im Modul *RoutingTable6*) wird daher für die IP-Adresse fe80::aaa:ff:fe00:2 der Interface-Wert „*-1 (unspec)*“ festgelegt. Die Prozedur endet mit der ICMPv6-Nachricht *DestUnreachable,* welche an das Modul *IPv6ErrorHandling* gesendet wird.

Durch diese aufgetretenen Probleme ist eine Fortführung des Testszenarios ohne Modifikation des INET-Quellcodes nicht möglich, da selbst im lokalen Subnetz keine IPv6-Pakete verschickt werden können.

6. Fazit

OMNeT++ bietet mit dem INET-Framework eine mächtige Möglichkeit Kommunikationsnetze zu simulieren. Ist man mit OMNeT++ selbst als Simulationswerkzeug vertraut, gestaltet sich die Arbeit mit dem INET-Framework als sehr einfach. Der aktuelle Implementierungsstand von IPv6 im INET-Framework ist allerdings selbst für simpelste Testszenarien unbrauchbar. Dies wurde anhand eines Szenarios demonstriert, bei dem Ping-Nachrichten zwischen zwei Hosts ausgetauscht werden sollten. Die Umsetzungsprobleme sind vor allem auf die mangelnden Konfigurationsmöglichkeiten von Interfaces und sonstigen IPv6-Parametern (z.B. Routing-Tabelle, IPv6-Präfixe, etc.) zurückzuführen. Eine wichtige Funktion, die im aktuellen INET-Release nicht umgesetzt wurde (aber vorgesehen ist), ist die Konfiguration von IPv6 Parametern über eine externe XML-Datei. Diese Möglichkeit war im Vorgängerprojekt IPv6Suite geboten und zeigt, dass viele der aufgetretenen Probleme dadurch gelöst werden könnten. Daneben bedarf es weiterer Anpassungen, die für eine vollständige IPv6-Funktionsfähigkeit bearbeitet werden müssen. Dazu gehören unter anderem die Stateless Address Autoconfiguration, die Bearbeitung der Routing-Tabelle oder aber die Implementierung von weiteren IPv6-Headern (z.B. *Authentication-Header* oder *DestinationOptionHeader*). Durch eine mangelnde Dokumentation in den überwiegenden Teilen von IPv6 gestaltet sich der Einstieg in die Abläufe und Module als sehr mühsam. Einzige Anlaufstelle zur Informationsbeschaffung ist der Quellcode der IPv6-Module. Ein äußerst hilfrei-

ches Werkzeug bei der Analyse von Simulationen bietet OMNeT++ mit der grafischen Benutzerschnittstelle *TKENV*. Durch das schrittweise Durchlaufen einer Simulation, können selbst kleineste Aktionen detailliert nachvollzogen werden. Dadurch konnten problematische Stellen beim Ablauf der IPv6-Testsimulation identifiziert werden.

7. Quellen

[1] **OMNeT++ Homepage**, Online: http://www.omnetpp.org [Aufruf im Januar 2012]

[2] A. Varga und R. Horning: „***An overview of the omnet++ simulation environment***“, Online: http://omnetpp.org/doc/workshop2008/omnetpp40-paper.pdf [Aufruf im Januar 2012]

[3] A. Varga und OpenSim Ltd: „***OMNeT++ User Manual - Version 4.2***“, Online: http://www.omnetpp.org/doc/omnetpp/Manual.pdf [Aufruf im Januar 2012]

[4] **INET-Framework Homepage**, Online: http://inet.omnetpp.org/ [Aufruf im Januar 2012]

[5] M. Tüxen, I. Rüngeler und E. P. Rathgeb: „***Interface connecting the INET simulation framework with the real world***“, Online: http://dl.acm.org/citation.cfm?id=1416267&dl=ACM&coll=DL&CFID=62068841&CFTOKEN=96347044 [Aufruf im Januar 2012]

[6] **INET Netzdokumentation**, Online: http://inet.omnetpp.org/doc/INET/neddoc/index.html [Aufruf im Januar 2012]

[7] **IPv6Suite Homepage**, Online: http://ctieware.eng.monash.edu.au/twiki/bin/view/Simulation/IPv6Suite [Aufruf im Januar 2012]

[8] **INET Changelog**, Online: http://inet.omnetpp.org/doc/INET/doxy/whatsnew.html [Aufruf im Januar 2012]

[9] **xMIPv6 github**, Online: https://github.com/zarrar/xMIPv6/ [Aufruf im Januar 2012]

[10] **IPv4 Routing-File**, Online: http://inet.omnetpp.org/doc/INET/neddoc/irt.html [Aufruf im Januar 2012]

[11] J. Lai: „**Introduction to IPv6 Simulation**“, Online: http://caia.swin.edu.au/talks/CAIA-TALK-040901A.pdf [Aufruf im Januar 2012]

[12] RFC 4862: „**IPv6 Statless Address Autoconfiguration**“. Online: http://www.ietf.org/rfc/rfc4862.txt, Internet Engineering Task Force (IETF), September 2007

[13] A. Badach und E. Hoffmann: „**Technik der IP-Netze**“, 2. Auflage, Carl Hanser Verlag, München 2007