Netzwerksimulation

Projekt im Rahmen der VL „Grundlagen der Rechnerkommunikation“

Projektgruppe: Bort

Volker Grabsch Martin Schröder

18. Februar 2008

Dieses Dokument beschreibt im Folgenden die grundlegenden Prinzipien und Funktionsweisen des Projektes für die Vorlesung „Grundlagen der Rechnerkommunikation“ des Wintersemesters 07/08 der Projektgruppe „Bort“. Dies umfasst eine kurze Beschreibung des benutzten Frameworks, gefolgt von der abstrakten Darstellung der verwendeten Algorithmen, geordnet nach den jeweiligen Netzwerkschichten.

Inhaltsverzeichnis

1 Framework

Für das Projekt wurde das C++ Simulationsframework OmNet++ in der Version 3.2 verwendet[1]. Dabei handelt es sich um einen objektorientierten, modularen Netzwerksimulator mit diskretem Ereignismodell[2]. Für die Zwecke unseres Projektes wurden nur die folgenden Eigenschaften des Frameworks verwendet:

1. Aufbau eines Graphen mit gerichteten Kanten, d.h. Simplexverbindungen zwischen einzelnen Knoten. Dies wird als Abstraktion für Netzwerkschichten, Computer, usw. genutzt.
2. Transport einer abstrakten Nachricht entlang genau einer Kante / Simplexverbindung, bzw. an den Knoten selbst, u.U. mit Zeitverzögerung.
3. Anhängen von C++ Objekten an diese Nachrichten als Abstraktion für den Inhalt eines Datenpakets.
4. Einlesen von Parametern aus der Umgebung in Form von primitiven Datentypen bzw. XML-Dateien.

All diese Eigenschaften wurden komplett in der Klasse `Layer` gekapselt[3], womit sich die eigentliche implementierte Funktion der Netzwerkschichten komplett von OmNet++ trennen lässt. Die einzige Ausnahme hiervon sind einige wenige Administrationsfunktionen, die von Omnet vorausgesetzt werden, die aber die Funktion weder beeinflussen noch begünstigen.
Darauf aufbauend werden durch die Klasse `Layer` die folgenden Funktionalitäten implementiert und für die erbenden Klassen bereitgestellt:

1. Duplexverbindungen zwischen Layern und Knoten
2. numerischer Zugriff auf diese Verbindungen
3. senden und empfangen von abstrakten Paket-Objekten[4] entlang der Verbindungen zur nächsthöheren bzw. -niedrigeren Schicht
4. erstellen, abbrechen und empfangen von Timeouts

Die Klasse `Packet` wird von allen verwendeten Paketklassen der einzelnen Netzwerkschichten geerbt und implementiert die folgenden, von OmNet++ völlig unabhängigen Funktionen:

1. Speichern von Ziel- und Quelladdresse sowie Paketgröße als primitiver numerischer Wert.
2. Speichern eines Pakettyps zur einfachen Identifikation der Art des jeweiligen Pakets.

[1] Siehe: `http://www.omnetpp.org/`
[2] Wortlaut der OmNet++ Dokumentation entnommen.
[3] Siehe Dateien `Layer.cc` und `Layer.hpp`.
[4] Objekten, die die Klasse `Packet` implementieren.

3. Speichern der Referenz auf ein weiteres `Packet`-Objekt als Abstraktion der mehrfachen Verschachtelung von Paketen.

2 Netzwerkschichten

Die Darstellung der Implementation der einzelnen Netzwerkschichten gliedert sich in jeweils zwei Unterabschnitte – eine Beschreibung des verwendeten Paketformats und eine Beschreibung der Abläufe in der jeweiligen Schicht. Eine grobe Übersicht kann man den Flussdiagrammen im letzten Abschnitt entnehmen.

Wie man darin sieht, sind die einzelnen Schichten strikt linear ausgelegt, d.h. Pakete werden nur von einer Schicht an die entweder nächsthöhere, oder -tiefere übergeben. Dabei werden unbekannte bzw. unerwartete Pakete einfach nach nach „oben“ oder „unten“ weitergereicht, was es technisch gesehen ermöglicht einen einzelnen PC sowohl als Nameserver, Standard-Gateway, Router und Dateidienst/-clienten zu benutzen. In den implementierten Beispielen werden jedoch die einzelnen Rechner nur, wie in der Projektaufgabenstellung gefordert, in einer einzigen Rolle benutzt.

2.1 Bitübertragungs-/Sicherungsschicht

Als physikalische Schicht wurden ein an Token-Ring angelehntes Protokoll implementiert. Wie in der Aufgabenstellung gefordert sind dadurch sowohl ein Token-Ring-Zugriff als auch eine Punkt-zu-Punkt-Verbindung unter Umgehung des Token-Ring-Protokolls möglich.

2.1.1 Token-Ring – Paketformat

Das von unserem Token-Ring-Protokoll verwendete Paketformat weist die folgenden Felder auf. Dabei werden die nicht fett gedruckten Felder aus der allgemeinen Packetklasse bezogen, die somit nicht direkt in den angegebenen Quelldateien vorkommen, sondern aus der Klasse `Packet` stammen und nur in der angegebenen Art und Weise benutzt werden:

1. Hardwareadresse des Quelladapters (im Folgenden Quell-MAC genannt)
2. Hardwareadresse des Zieladapters (im Folgenden Ziel-MAC genannt)
3. Paketgröße, Kopf + angehängte Daten (Pakete der höheren Schichten)
4. **Token-Ring Frame Typ**, legt fest ob es sich um ein Token- oder Daten-Frame handelt.

Der Token-Ring-Pakettyp wird in den Datei `TokenRingPacket.hpp` implementiert.

2.1.2 Token-Ring – Schicht

Das verwendete Protokoll reagiert wie folgt auf von außen ankommende Pakete:

1. Zuerst wird getestet, ob es sich überhaupt um ein Token-Ring Paket handelt. Wenn nicht, wird es einfach unbearbeitet an die nächsthöhere Schicht weitergereicht.
2. Es wird der Pakettyp bestimmt, d.h. ob es ein Token- oder Daten-Frame ist.
3. Wenn es sich um ein Token-Frame handelt,
   a) wird aus dem Sendepuffer ein Paket entnommen, (Sollte der Puffer leer sein, so wird das Token wieder nach unten, in diesem Fall auf das virtuelle Kabel, gesendet.)
   b) wird ein internes Flag gesetzt, dass diese Schicht im Besitz des Tokens ist, und
   c) wird das entnommene Paket in ein Data-Frame verpackt und anstelle des Tokens nach unten gesendet.
4. Wenn es sich um ein Daten-Frame handelt:
   a) Wenn die Quell-MAC des Pakets diesem Adapter entspricht und das interne Token-Flag gesetzt ist[5], so wird das Flag zurückgesetzt und ein neues Token nach unten gesendet.
   b) Wenn die Ziel-MAC des Pakets diesem Adapter entspricht, so wird das Paket entpackt und das enthaltene Paket, falls vorhanden, an die nächste Schicht weitergereicht.
   c) Wenn keins von beidem der Fall ist, wird das Daten-Frame unverändert nach unten gereicht.

Wenn das Paket jedoch von einer höheren Schicht kommt, so wird es einfach in einen dafür vorgesehenen Datenpuffer mit FIFO Zugriff kopiert.

Die Implementation des beschriebenen Protokolls erfolgt in den Dateien `TokenRingLayer.hpp` und `TokenRingLayer.cc`.

Es sei zudem angemerkt, dass sich dieses Protokoll nicht um die initiale Einspeisung des ersten Tokens, dessen Wiederherstellung im Fehlerfall oder der Ringbildung kümmert. Diese Funktionalität wird jeweils durch einen als „Token-Ring-MAU“ bezeichneten Hub erledigt, an den ein jeder Token-Ring Adapter angeschlossen werden muss. Dieser ist in den Dateien `TokenMAULayer.hpp` und `TokenMAULayer.cc` implementiert.

[5] Eigentlich wäre es nur notwendig auf den Besitz des Tokens zu testen, der gewählte Weg hat aber diagnostische Vorteile.

2.2 Vermittlungsschicht

Die Vermittlungsschicht wurde von uns in zwei Hälften geteilt.

Die erste, die Gateway-Schicht, ist für die Auflösung von Netzwerk-Adressen nach (rein lokalen) MAC-Adressen zuständig. Im Folgenden nennen wir Netzwerk-Adressen der Kürze halber „IPs“. Die Gateway-Schicht entscheidet, ob ein Paket für den Host selbst bestimmt war, oder an den Standard-Gateway zur erweiterteten Wegewahl weitergeleitet werden muss. Sie nimmt somit eine zu ARP und IP ähnliche Rolle ein.

Die zweite Hälfte besteht aus der Routing-Schicht. Sie leitet die von oben und unten ankommenden Pakete dem jeweils korrekten nächsten Router oder dem tatsächlichen Ziel zu. Außerdem wird in ihr ein zuschaltbares Routing-Protokoll implementiert, welches Routing-Informationen sammelt und an die benachbarten Router verteilt.

2.2.1 Gateway – Paketformat

Das von unserer Gateway-Schicht verwendete Paketformat weist die folgenden Felder auf, wobei die nicht fett gedruckten Felder erneut aus der allgemeinen Packetklasse bezogen werden:

1. Hardwareadresse des Quelladapters (im Folgenden Quell-MAC genannt)
2. Hardwareadresse des Zieladapters (im Folgenden Ziel-MAC genannt)
3. **Netzwerkadresse des Quelladapters** (im Folgenden Quell-IP genannt)
4. **Netzwerkadresse des Zieladapters** (im Folgenden Ziel-MAC genannt)
5. Paketgröße, Kopf + angehängte Daten (Pakete der höheren Schichten)
6. **Keepalive-Flag**, legt fest ob das Paket regulär ist, oder ob es sich nur um die Information an andere handelt, dass die Schicht / der Knoten noch aktiv ist.

Der Gateway Pakettyp wird in den Datei `GatewayPacket.hpp` implementiert.

2.2.2 Gateway – Schicht

Das verwendete Protokoll reagiert wie folgt auf ein von einer niedrigeren Schicht ankommendes Paket:

1. Zuerst wird anhand des Pakettyps getestet, ob das Paket ein Gateway-Paket ist. Wenn nicht, wird es unverändert und direkt an die nächsthöhere Schicht weitergeleitet.
2. Sonst wird getestet, ob es sich bei den Paket um ein Keep-Alive handelt. Wenn ja, so wird die interne MAC/IP Tabelle mit den Daten des empfangenen Paketes ergänzt und das Paket verworfen.

3. Handelt es sich hingegen um ein reguläres Paket, entscheidet die Ziel-IP über die weitere Vorgehensweise:
   a) Ist sie entweder eine Broadcast-IP oder die eigene IP, so wird das Paket entpackt, und der Inhalt, sofern vorhanden, an die nächste Schicht gesendet.
   b) Ist die eingetragene IP des Standard-Gateways ungleich 0, wird der Inhalt des Pakets in ein neues Gateway-Paket verpackt, und nach unten an die MAC-Adresse des Standard-Gateways weitergereicht.
   c) Wenn kein Standard-Gateway definiert wurde (also dessen IP gleich 0 ist), wird das Paket entpackt und nach „oben“ weitergereicht, um vom Routing-Layer behandelt zu werden.

Kommt das Paket jedoch von einer höheren Schicht, so wird versucht, die Ziel-IP des Pakets zu einer Ziel-MAC aufzulösen. Dabei wird eine Broadcast-IP durch eine Broadcast-MAC aufgelöst. Wenn das IP/MAC-Paar aufgelöst werden konnte, so wird das Paket mit diesen Daten nach „unten“ gesendet. Konnte kein passendes IP/MAC-Paar gefunden werden, so wird das Paket an die MAC des Standard-Gateways gesendet. Sollte kein Standard-Gateway definiert sein, wird das Paket verworfen.

Die IP/MAC-Tabelle wird durch die von anderen Knoten empfangenen als Keep-Alive markierten Gateway-Pakete befüllt. Ein solches Paket wird von einer jeden Gateway-Schicht in regelmäßigen Abständen gesendet. Die Tabelle wird dabei in gewissen regelmäßigen Zyklen von alten Einträgen befreit.

Die Gateway-Schicht wird in den Dateien `GatewayLayer.hpp` und `GatewayLayer.cc` implementiert.

2.2.3 Routing – Paketformat

Das von unserem Routing-Protokoll zur Kommunikation zwischen den Routern verwendete Paketformat weist die folgenden Felder auf:

1. Netzwerk-Adresse des Quelladapters (im Folgenden Quell-IP genannt)
2. Netzwerk-Adresse des Zieladapters (im Folgenden Ziel-IP genannt)
3. Paketgröße, Kopf + angehängte Daten (Pakete der höheren Schichten)
4. **Routingtabelle**, enthält eine Kopie der Routingtabelle eines Routers.
5. **Dirty-Flag**, zeigt an, ob sich die übergebene Routingtabelle seit der letzten Kommunikation verändert hat.

Pakete die nur geroutet werden, werden in ein Standardpaket verpackt, wobei die Addressfelder Netzwerk-Addressen sind, in denen das nächste benötigte Zwischenziel und die Ursprungsquelle eingetragen wird.

Der Routing-Pakettyp wird in den Datei `RoutingPacket.hpp` implementiert.

2.2.4 Routing – Schicht

Das von uns für die Router-zu-Router Kommunikation eingesetzte Routingprotokoll basiert im Wesentlichen auf einem Distance-Vector-Verfahren. Dabei gilt die Latenz zum nächsten Router in der Kette vom momentanen Router zum Ziel als Maß für die Distanz. Mit anderen Worten: Der erste Router, der Informationen über einen Knoten liefert, wird in der Tabelle gespeichert. Informationen von anderen oder über andere Router werden ignoriert, bis der gegebene Eintrag in einer gewissen Zeit nicht mehr erneuert wird[6]. Die Routingtabelle enthält zudem auch die IPs aller lokal an den Router angeschlossenen Rechner, wobei bei einem IP-Adresskonflikt stets die lokale IP bevorzugt wird.

Um die IP-Adressen der lokalen Knoten herauszufinden, sendet die Routing-Schicht, sofern das oben genannte Routingprotokoll aktiviert ist, im regelmäßigen Abstand Echo-Pakete an die Broadcast-IP über alle vorhandenen Ausgänge, die in eine tiefere Schicht führen. Ein Knoten mit nicht aktivierter Routingschicht antwortet mit einem eigenen als Antwort markiertem Echo-Paket an die IP des Knotens, der das Echo versendet hat. Diese Echo-Pakete sind um einzelnes Feld erweiterte Standardpakete, wobei die Adressen Netzwerkadressen (IPs) sind. Die Erweiterung ist ein einziges Flag, das identifiziert, ob das Paket eine Echo-Anfrage oder Echo-Antwort ist. Der Router nimmt alle Antworten auf das Echo als lokale Adressen in seine Routingtabelle auf.

Empfängt jedoch ein anderer Router eine Echo-Anfrage, so versendet er zusätzlich noch ein oben beschriebenes Routing-Paket, welches eine Kopie seiner eigenen Routingtabelle enthält. Diese Tabelle wird von dem Urpsrungs-Router verarbeitet, indem alle noch unbekannten Adressen zur eigenen Tabelle hinzugefügt werden. Die vorhandenen Adressen, die mit dem gespeicherten physischen Ausgang verbundenen sind, werden so markiert, dass sie nicht beim nächsten Löschvorgang entfernt werden.

Auf diese Art und Weise wird sichergestellt, dass sich Änderungen schrittweise durch das gesamte Netz verteilen. Um einen guten Kompromiß aus Änderungsgeschwindigkeit und Netzbelastung zu erreichen, wird die Wartezeit zwischen dem Aussenden einer neuer Reihe an Echo-Paketen bei vorgenommenne Änderungen auf ein gewisses Minimum reduziert. Bei ausbleibender Änderung wird die Wartezeit exponentiell bis zu einem gewissen Maximum erhöht. Dieses Übertragen der Änderungen erfolgt durch Setzen des „Dirty-Flags“ in den besprochenen Routing-Paketen.

Wenn das eigentliche Routingprotokoll abgeschaltet wurde, so verhält sich die Routingschicht bei einem eintreffenden Paket wie folgt:

[6] Dies unterscheidet den benutzten Algorithmus von „klassischem“ Distance-Vector-Routing, da stets nur eine Verbindung pro Ziel gespeichert wird, und zwar die heuristisch beste.

1. Kam das Paket von einer höheren Schicht, wird es einfach an die nächstniedrigere Schicht weitergeleitet, da angenommen wird, dass in diesem Fall ein lokaler Standard-Gateway eingetragen wurde, oder das Ziel lokal angeschlossen ist. Die Ausnahme davon bilden Pakete, die an die eigene IP adressiert sind. Diese werden direkt zurück an die höheren Layer geleitet.
2. Anderenfalls, wenn es sich um ein Echo-Paket handelt, wird ein als Antwort markiertes eigenes Echo-Paket zurückgesendet.
3. In den übrigen Fällen wird das Paket gegebenenfalls entpackt[7] und an die nächsthöhere Schicht weitergeleitet.

Wenn das Routingprotokoll angeschaltet wurde, wird dieses Verhalten so verändert, dass die Routingtabelle gefüllt wird und ein direktes Routing der eingehenden Pakete stattfindet. So wird die Kommunikation zwischen den Routern bzw. zwischen Router und einzelnen lokalen Knoten ermöglicht. Dabei verhält sich diese Schicht beim Empfang von Paketen wie folgt:

1. Wenn das Paket von einer höheren Schicht ankommt, wird die IP des übergebenen Pakets in der Routingtabelle nachgeschlagen. Falls ein Eintrag gefunden wird, wird das Paket mit einem anderen Paket umhüllt, dessen Ziel-IP aus der Routingtabelle entnommen wurde und dessen Quell-IP die IP des momentanen Routers ist. Danach wird das Paket über den ebenfalls der Tabelle entnommenen physikalischen Ausgang an eine niedrigere Schicht gesendet. Die Ausnahme sind erneut Pakete, die an die eigene IP adressiert sind. Diese werden sofort an die höheren Schichten zurückgeleitet.
2. Wenn das Paket von einer niedrigeren Schicht kommt, wird zunächst der Typ des Pakets festgestellt.
3. Wenn es sich um ein Echo Paket handelt:
   a) Wenn das Paket eine Echo-Anforderung ist, so wird ein Antwort-Echo-Paket zurückgesendet.
   b) Direkt danach wird ein Routing-Paket mit einer Kopie der momentanen Routingtabelle an die Quell-IP des Echo Pakets gesendet.
   c) Handelte es sich aber um eine Antwort auf ein von dieser Schicht ausgehendes Echo Paket, so wird die Quell-IP als lokaler Knoten in die Routingtabelle übernommen.
4. Wenn es sich um ein Routing-Paket handelt:
   a) Wenn das Paket nicht von der Routing-Schicht selbst stammt[8], wird die im Paket enthaltene Routingtabelle entpackt.

[7] Um die IPs der Originalquelle und des Originalziels nicht überschreiben zu müssen, wird beim Routing das Paket in ein Standardpaket mit veränderten Adressen verpackt.

[8] Dies kann passieren, wenn sie mit zwei unterschiedlichen physikalischen Ausgängen an dasselbe Netz angeschlossen ist.

b) Es werden alle Einträge als nicht-lokale IPs übernommen, die auf eine bislang unbekannte IP zeigen. Der dazugehörige physikalische Ausgang ist derjenige, über den das Routing-Paket ankam. Als dazugehöriger Router wird die Quell-IP des Pakets angenommen.

c) Bereits bekannte Einträge werden, wenn ihr physikalischer Ausgang und zugehöriger Router zu den Daten im Paket passen, so markiert, dass sie nicht beim nächsten Löschvorgang entfernt werden.

d) Wenn sich irgendetwas an der Routingtabelle geändert hat, wird die eigene Tabelle als „dirty“ markiert.

5. Falls es sich um einen anderen Pakettyp handelt:

   a) Pakete an Broadcast-IPs werden ignoriert und fallen gelassen. Dies sorgt für eine Abtrennung der Netze in unterschiedliche Broadcast-Domänen.

   b) Zunächst wird ein gegebenenfalls vorhandener Routing-Wrapper entfernt, um die echte Ziel-IP des Pakets in Erfahrung zu bringen.

   c) Wenn diese IP mit der des Routers übereinstimmt, so wird das Paket einfach an die nächsthöhere Schicht weitergeleitet.

   d) Sonst wird angenommen, dass das Paket geroutet werden muss.

   e) Dazu wird die Ziel-IP in der Routingtabelle nachgeschlagen und, sofern gefunden, verpackt und an die dort eingetragene IP über den eingetragenen Ausgang an eine niedrigere Schicht übergeben.

   f) Wenn die IP nicht aufgelöst werden kann, wird das Paket einfach ignoriert und fallen gelassen.

Auf diese Weise werden Pakete geroutet, und die routerinterne Kommunikation abgesichert. Aufgrund der Verwendung eines Distance-Vector-ähnlichen Algorithmus werden Änderungen durch das ganze Netz verteilt, bis sich nach einiger Zeit eine Stabilisierung[9] einstellt. Die Optimalität des Pfades kann dabei jedoch nicht garantiert werden, da die Abschätzung durch die Latenz zum nächsten Sprung zu gierig (greedy) ist, und sich somit in lokalen Optima verfangen kann. Schleifenlosigkeit wird nach einer Änderung unter Umständen erst im Stabilitätspunkt wiederhergestellt. Da das Protokoll keine Lebensdauer eines individuellen Pakets vorsieht kann es somit zu langen Paketschleifen kommen, die sich aber ebenso spätestens im Stabilitätspunkt auflösen.

Die Routing-Schicht wird in den Dateien `RoutingLayer.hpp` und `RoutingLayer.cc` implementiert.

[9] Dass stets eine Stabilisierung eintritt kann hier nur aufgrund von empirischer Erfahrung mit dem Algorithmus angenommen werden.

2.3 Transportschicht

Die Transportschicht stellt bei uns die korrekte Reihenfolge der Pakete sicher, garantiert ein Ankommen der Pakete, sofern es überhaupt eine Route zum Ziel gibt, und schützt durch eine einfache statische Flusssteuerung vor Überlastungen. Die korrekte Fragmentierung und Reassemblierung der Pakete wurde aus der eigentlichen Transportschicht ausgelagert, und wird von einer direkt über ihr stehenden, eigenen Schicht erledigt. Dies trennt auf natürliche Weise diese beiden Teilgebiete. Im Folgenden wird nur die Transportschicht näher beleuchtet, da die Fragmentierungsschicht eher simpel und wenig funktionell wenig erhellend ist. Es dürfte ausreichen zu sagen, dass sie davon ausgehen kann, dass die Pakete, dank der Transportsicherung, in der richtigen Reihenfolge und stets komplett ankommen, so dass die einzige Aufgabe das Verbinden und Trennen von Paketen ist. Dies geschieht durch Nutzung fortlaufender Sequenznummern pro Verbindung für jedes aufzuteilende Paket.

2.3.1 Transport – Paketformat

Das von unserem Transportprotokoll verwendete Paketformat weist die folgenden Felder auf:

1. Netzwerkadresse des Quelladapters (im Folgenden Quell-IP genannt)
2. Netzwerkadresse des Zieladapters (im Folgenden Ziel-IP genannt)
3. Paketgröße, Kopf + angehängte Daten (Pakete der höheren Schichten)
4. **Pakettyp**, zeigt an um welche Art von Paket es sich handelt[10]
5. **Sequenznummer der Quelle**, speichert die Sequenznummer, die dieses Paket beim Absenden von ihrer Quelle zugewiesen bekommen hat.
6. **Zusatzsequenznummer 1 & 2**, speichern zwei weitere Sequenznummer für administrative Zwecke[11]

Der Pakettyp der Transportschicht wird in der Datei `TransportPacket.hpp` implementiert.

2.3.2 Transport – Schicht

Das von uns zur Transportsicherung verwendete Protokoll ist verbindungsorientiert und nutzt zum Verbindungsaufbau und -abbau einen 3 Wege Handshake. Die Fehlerbehandlung erfolgt durch eine leicht modifizierte Variante des „Selective-Repeat“, bei der ein NACK so umgedeutet wurde, dass damit der erfolgreiche Empfang eines Paketes bestätigt wird, das nicht in korrekter Reihenfolge ankam. Somit werden fehlende Pakete schon vor dem Timeout erkannt und können. Die Flusssteuerung wird mittels eines Gleitenden Fens-

[10] Die möglichen Arten werden in Abschnitt 2.3.2 genauer erläutert.
[11] Details in Abschnitt 2.3.2

ters (Sliding Window) implementiert, wobei die größe des erlaubten Fensters statisch vordefiniert wird.

Bei einem Verbindungsaufbau verhält sich die Transportschicht wie folgt, wobei P1 und P2 jeweils die beiden kommunizierenden Transportschichten sind, und die Verbindung von P1 ausgeht.

1. P1 generiert eine zufällige Sequenznummer $S1$ und erstellt damit einen Verbindungseintrag in seiner Verbindungstabelle mit der Flussrichtung $P1 \rightarrow P2$ und der Sequenznummer $S1 - 1$[12].
2. P1 sendet ein SYN Paket an P2, wobei das Quellsequenzfeld mit der generierten Sequenznummer $S1$ gefüllt wird.
3. P2 empfängt das SYN Paket und erstellt zwei Verbindungseinträge $P1 \rightarrow P2$ und $P2 \rightarrow P1$.
4. Der ersten der beiden Verbindungen weist P2 die Sequenznummer $S1$ aus dem SYN Paket zu. Für die zweite Verbindung generiert P2 eine neue Sequenznummer $S2$ und weist der Verbindung $S2 - 1$ als letztes bestätigtes Paket zu.
5. P2 sendet ein SYN+ACK Paket an P1, wobei das Quellsequenzfeld S2 trägt, und das 1. Zusatzsequenzfeld nochmals $S1$ enthält.
6. P1 empfängt das SYN+ACK von P2, aktiviert die Verbindung $P1 \rightarrow P2$ sofern $S1$ korrekt zurückübertragen wurde, und erhöht die Sequenznummer der Verbindung um 1. Danach wird die Verbindung $P2 \rightarrow P1$ mit $S2$ angelegt und aktiviert. P1 sendet dann ein ACK zurück an P2, wobei die Quellsequenz der Verbindung $P1 \rightarrow P2$ entnommen wurde und $S2$ im Extrasequenzfeld gespeichert wird.
7. P2 empfängt das ACK und aktiviert daraufhin, sofern die Sequenznummern passen und korrekt hochgezählt wurden, beide Verbindungen.

Auf diese Art werden werden bei beiden Teilnehmern die gleichen Verbindungen mit identischen Sequenznummern erstellt, und eine Datenkommunikation kann begonnen werden. Es sei hierbei angemerkt, dass ein jeder Verbindungseintrag genau zwei Sequenznummern speichert: Die Sequenznummer des zuletzt quittierten Pakets und die Sequenznummer die das nächste Paket bekommen muss. Eine graphische Übersicht findet sich in bei den Flußdiagrammen auf Seite 17 in Abbildung 1, wobei darauf zu achten ist, dass hierbei pro Verbindung nur die Sequenznummer des zuletzt quittierten Pakets angezeigt wird.

Beim Datenaustausch wird die folgende Abfolge eingehalten. Hierbei wird erneut ein Datenfluss von P1 nach P2 angenommen.

1. P1 testet ob die Verbindung nach P2 steht und aktiv ist.
2. P1 erstellt ein Datenpaket mit der nächstmöglichen Sequenznummer der

[12]Diese Nummer zeigt an, welches Paket als letztes korrekt bestätigt wurde. In diesem Fall noch keins.

Verbindung $P1 \rightarrow P2$, sendet dieses an P2 und setzt einen Timeout für das erneute versenden.

3. P2 empfängt das Datenpaket. Falls es das Paket nicht empfängt läuft bei P1 der Timeout ab und das Paket wird erneut gesendet. So kommt es irgendwann bei P2 an.
   a) Falls das Datenpaket mit der richtigen (als nächstes für die Verbindung $P1 \rightarrow P2$ erwarteten) Sequenznummer ankam, sendet P2 ein ACK an P1, wobei das Quellsequenzfeld mit der nächstmöglichen Nummer aus der Verbindung $P2 \rightarrow P1$, und das Extrasequenzfeld mit der Sequenznummer des Datenpakets gefüllt wird. Danach wird das Datenpaket entpackt und an die nächsthöhere Schicht geleitet.
   b) Falls das Datenpaket nicht mit der richtigen Sequenznummer ankam, wird es entweder verworfen, wenn es ausserhalb des Sliding Window ist, oder es wird durch ein NACK bestätigt. Das NACK funktioniert dabei wie ein ACK, nur dass das zweite Extrasequenzfeld mit der eigentlich für die Verbindung $P1 \rightarrow P2$ erwarteten Sequenznummer befüllt wird. Das Paket wird dann solange zwischengepuffert, bis die Reihenfolge durch den Empfang der erwarteten Datenpakete wiederhergestellt wurde. Erst dann wird es entpackt und an die nächsthöhere Schicht geleitet.
4. P1 empfängt nun wiederum ein ACK oder ein NACK
   a) Bei einem ACK wird der Timeout für das zum ACK gehörende Datenpaket entfernt.
   b) Bei einem NACK wird ebenso der Timeout für das zum NACK gehörende Datenpaket entfernt, aber zusätzlich werden noch alle Pakete erneut gesendet (und deren Timeouts neu gestartet), die zwischen dem durch NACK bestätigten Paket liegen, und dem Paket, auf dass im zweiten Extrasequenzfeld des NACK verwiesen wird. Dabei werden all' jene dazwischenliegenden Pakete ignoriert, die bereits selbst ein NACK bzw. ein ACK erhalten haben.

Am Ende dieser Abfolge ist die Reihenfolge und jedes zwischendurch verlorengegangene Paket wiederhergestellt. Es sei angemerkt, dass falls ACK und NACK Pakete verloren gehen, diese erst nach erneuter Ankunft des zu ihnen gehörigen Datenpakets neu gesendet werden, während Datenpakete entweder durch einen Timeout oder, oftmals schneller, als Nebeneffekt eines NACK neu gesendet werden.

Ein Verbindungsabbau erfolgt erneut durch einen 3-Wege Handshake und wird ausgelöst, wenn ein Timeout für die jeweilige Verbindung erreicht wird und bei einem anschließendes Test festgestellt wird, dass die Verbindung inaktiv ist, also weder auf Pakete wartet, noch Pakete senden will. Dabei werden nur Verbindungen von der jeweiligen Transportschicht selbst zu anderen Transportschichten berücksichtigt, da die übrigen während des 3-Wege

Handshakes mit entfernt werden. Der Ablauf ist dann wie folgt, wobei erneut ein Ausgehen von P1 vorausgesetzt wird.

1. P1 erhält den Timeout für die Verbindung $P1 \rightarrow P2$ und erkennt, dass diese Verbindung inaktiv ist.
2. P1 sendet ein FIN Packet an P2 und setzt die Verbindung $P1 \rightarrow P2$ und $P2 \rightarrow P1$ auf „inaktiv“, so dass keine neuen Datenpakete mehr über diese Verbindung gesendet werden können, bis sie komplett abgebaut ist. Alte Pakete (wegen erneutem Senden) und ACKs/NACKs können aber immer noch gesendet werden, auch wenn dies aufgrund der Bedingungen and das Auslösen des FIN nicht auftreten sollte.
3. P2 empfängt das FIN Packet und inaktiviert die Verbindungen $P1 \rightarrow P2$ und $P2 \rightarrow P1$, wartet jedoch noch mit dem Bestätigen des FIN bis alle Pakete für die erste Verbindung bis zur Sequenznummer des FIN angekommen sind und die zweite Verbindung für alle Pakete ein ACK bzw. NACK erhalten hat.
4. P2 sendet ein FIN+ACK und setzt die Verbindungen $P1 \rightarrow P2$ und $P2 \rightarrow P1$ auf „tot“, wodurch alle weiteren Pakete abgelehnt werden und die Verbindung eventuell wieder neu aufgebaut werden kann.
5. P1 empfängt das FIN+ACK und setzt nunmehr die beiden Verbindungen $P1 \rightarrow P2$ und $P1 \rightarrow P2$ auf „tot“.

Am Ende dieser Abfolge sind die Verbindungen in beide Richtungen abgebaut, ohne dass ein Paket dabei verlorengegangen sein sollte.

Die Flusssteuerung erfolgt einfach und direkt durch Vorgabe eines Sliding-Window für die Auswahl an Sequnznummern pro Verbindung. Jede Verbindung darf dabei nur eine gewisse Menge an Datenpaketen mehr versenden, als bislang Pakete bestätigt wurden. Gleiches gilt für den Empfang, bei denen jedes Paket, das mit seiner Sequenznummer „zu weit in der Zukunft“ liegt einfach ignoriert wird. Bei letzterem stellt die gegenüberliegende Transportschicht ein erneutes Versenden des verworfenen Paketes sicher. Bei ersterem Fall, dass wir selbst mehr Pakete als erlaubt senden wollen, können wir die Pakete nicht ignorieren oder verwerfen, da in dem von uns implementierten Framework eine derartige Kommunikation zwischen den Schichten nicht möglich ist. Die übermäßig ankommenden Pakete müssen daher zwischengepuffert werden, und werden verarbeitet sobald die Verbindung wieder freie Sequenznummern übrig hat.

Die Transportschicht wird in den Dateien `TransportLayer.hpp` und `TransportLayer.cc` implementiert. Die Hilfsklassen für das Verwalten von Verbindungseinträgen sind in den Dateien `ConnTracker.hpp` und `ConnTracker.cc` implementiert.

2.4 Anwendungsschicht

In der Anwendungsschicht stehen zwei Dienste zur Verfügung, der Namensdienst und die Dateiübertragung. Jeder simulierte PC besitzt genau einen dieser beiden Dienste, mit Ausnahme der Router. Auf einem Router könnte zwar ebenfalls ein Dienst der Anwendungsschicht laufen, im Rahmen der Aufgabenstellung wird jedoch darauf verzichtet.

2.4.1 Namensdienst – Paketformat

Das von unserem Namensdienstprotokoll verwendete Paketformat weist die folgenden Felder auf:

1. Netzwerkadresse des Quelladapters (im Folgenden Quell-IP genannt)
2. Netzwerkadresse des Zieladapters (im Folgenden Ziel-IP genannt)
3. Paketgröße, Kopf + angehängte Daten (Pakete der höheren Schichten)
4. **Namen**, der Name der mit der gegebenen Adresse verbunden ist.
5. **Adresse**, die IP Adresse die mit dem gesuchten Namen verbunden ist.

Hierbei wird vom anfragenden PC der Name vorgegeben und die Adresse als Antwort vom Namensdienst eingetragen.

2.4.2 Namensdienst – Schicht

Der Ablauf der Kommunikation zwischen einem Clienten und dem Namensdienst läuft wie folgt ab:

1. Der Client bereitet ein Paket im Format des Namensdienstes vor, und trägt im Namensfeld den gesuchten Namen ein. Das Adressfeld kann beliebig gefüllt werden, es wird durch den Namensdienst ignoriert.
2. Der Namensdienst empfängt das Paket, sucht den Namen in seiner Namenstabelle und trägt, sofern der Name aufgelöst werden konnte, die aufgelöste Netwerkadresse und den Namen in ein neues Namenspaket ein und sendet es an den Client zurück. Wenn der Name nicht aufgelöst werden konnte, wird als Adresse eine -1 eingetragen.
3. Der Client empfängt das Paket und erfährt somit den passenden Namen zu der Adresse, wobei er das Adressfeld auf Existenz einer -1 testet und darauf entsprechend reagiert.

In der vorliegenden Implementation des Netzwerks und der Verteilung der Schichten erfolgt die Kommunikation zwischen den Clienten und dem Namensdienst auch über die Transportsicherung und läuft daher verbindungsorientiert ab.

2.4.3 Dateiübertragung – Paketformat

Die zur Dateiübertragung verwendeten Pakete weisen folgenden Felder auf:

1. Netzwerkadresse des Quelladapters (im Folgenden Quell-IP genannt)
2. Netzwerkadresse des Zieladapters (im Folgenden Ziel-IP genannt)
3. Paketgröße, Kopf + angehängte Daten (Pakete der höheren Schichten)
4. **Typ**, der angibt, welche Aktion auf dem Dateisystem auszuführen ist, bzw. um was für eine Antwort es sich handelt
5. **String-Parameter**, ein abstrakter Parameter für Aktionen und Antworten, die eine Zeichenkette als Parameter benötigen (z.B. Dateipfad)
6. **Int-Parameter**, ein abstrakter Parameter für Aktionen und Antworten, die eine Zahl als Parameter benötigen (z.B. Dateigröße)

Es gibt folgende **Typen**:

1. *CONN-REQ* – Anfrage zum Aufbau einer Verbindung
2. *CONN-ACK* – Antwort: Bestätigung der Verbindung, insbesondere die Bestätigung, dass sich am anderen Ende auch ein Dateiübertragungsdienst befindet
3. *CD-REQ*, *CD-ACK* und *CD-DENY* – Anfrage, Positiv-Antwort und Negativ-Antwort für die Aktion „Verzeichniswechsel“
4. *DEL-REQ*, *DEL-ACK* und *DEL-DENY* – Anfrage, Positiv-Antwort und Negativ-Antwort für die Aktion „Datei oder Verzeichnis löschen“
5. *DIR-REQ*, *DIR-ACK* und *DIR-DENY* – Anfrage, Positiv-Antwort und Negativ-Antwort für die Aktion „Verzeichnisinhalt anzeigen“
6. *MKDIR-REQ*, *MKDIR-ACK* und *MKDIR-DENY* – Anfrage, Positiv-Antwort und Negativ-Antwort für die Aktion „Verzeichnis erstellen“
7. *MKFIL-REQ*, *MKFIL-ACK* und *MKFIL-DENY* – Anfrage, Positiv-Antwort und Negativ-Antwort für die Aktion „Datei erstellen“
8. *GET-REQ*, *GET-ACK* und *GET-DENY* – Anfrage, Positiv-Antwort und Negativ-Antwort für die Aktion „Datei herunterladen“
9. *PUT-REQ*, *PUT-ACK* und *PUT-DENY* – Anfrage, Positiv-Antwort und Negativ-Antwort für die Aktion „Datei hochladen“

Das Paketformat der Dateiübertragungsschicht ist in der Datei `FilePacket-.hpp` implementiert.

2.4.4 Dateiübertragung – Schicht

Die Dateiübertragungsschicht ist Client und Server zugleich. Sie antwortet einerseits auf eingehende Anfrage-Pakete (Server) und stellt andererseits der Shell ihre Funktionen zur Verfügung (Client). So ist das Wissen über das Protokoll und Paketformat in der Schicht gekapselt. Die Außenwelt (Shell) bedient lediglich die bereitgestellten Funktionen.

Durch Shellkommando oder Anfrage-Paket ausgelöste lokale Aktionen werden direkt ausgeführt. Remote-Aktionen werden durch Versenden eines entsprechenden Anfrage-Pakets realisiert. Eingehende Antwortpakete werden dann dem Benutzer präsentiert.

Es werden jedoch keine echten Dateien übertragen. Sämtliche Aktionen finden in virtuellen Dateisystemen statt. Der Inhalt der Dateisysteme wird zu Beginn der Simulation pro PC durch entsprechende XML-Konfigurationsdateien festgelegt.

Dabei bildet das virtuelle Dateisystem nur Dateipfade und Dateigrößen ab. Mehr ist für die Simulation auch nicht notwendig: Große Dateien verursachen bei einer *GET-REQ*-Anfrage ein entsprechend großes *GET-ACK*-Antwortpaket, das von den tieferen Schichten entsprechend in kleinere Pakete zerlegt wird, die übertragen, in die richtige Reihenfolge gebracht und wieder zusammengefügt werden müssen.

Die Dateien und Verzeichnisse werden auf Paket-Ebene stets durch absolute Pfade adressiert. Jedoch akzeptieren die Client-Funktionen auch relative Pfade, die clientseitig in absolute Pfade umgewandelt werden.

Die Dateiübertragungsschicht wird in den Dateien `FileLayer.hpp` und `FileLayer.cc` implementiert. Das virtuelle Dateisystem wird in den Dateien `SimpleFS.hpp` und `SimpleFS.cc` implementiert.

Flussdiagramme

Zur besseren Veranschaulichung der inneren Abläufe in den verschiedenen Schichten folgen nun einige Flussdiagramme.

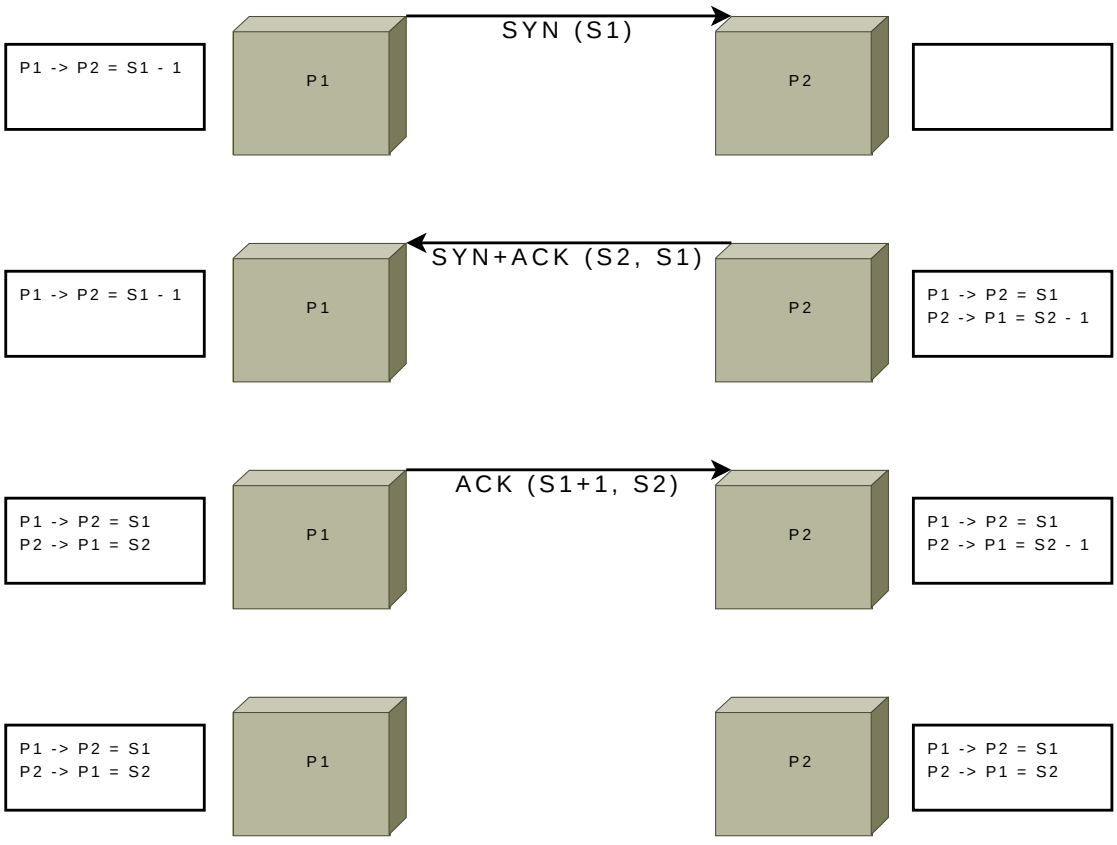

Abbildung 1: Ablauf eines 3-Wege Handshakes zum Verbindungsaufbau.

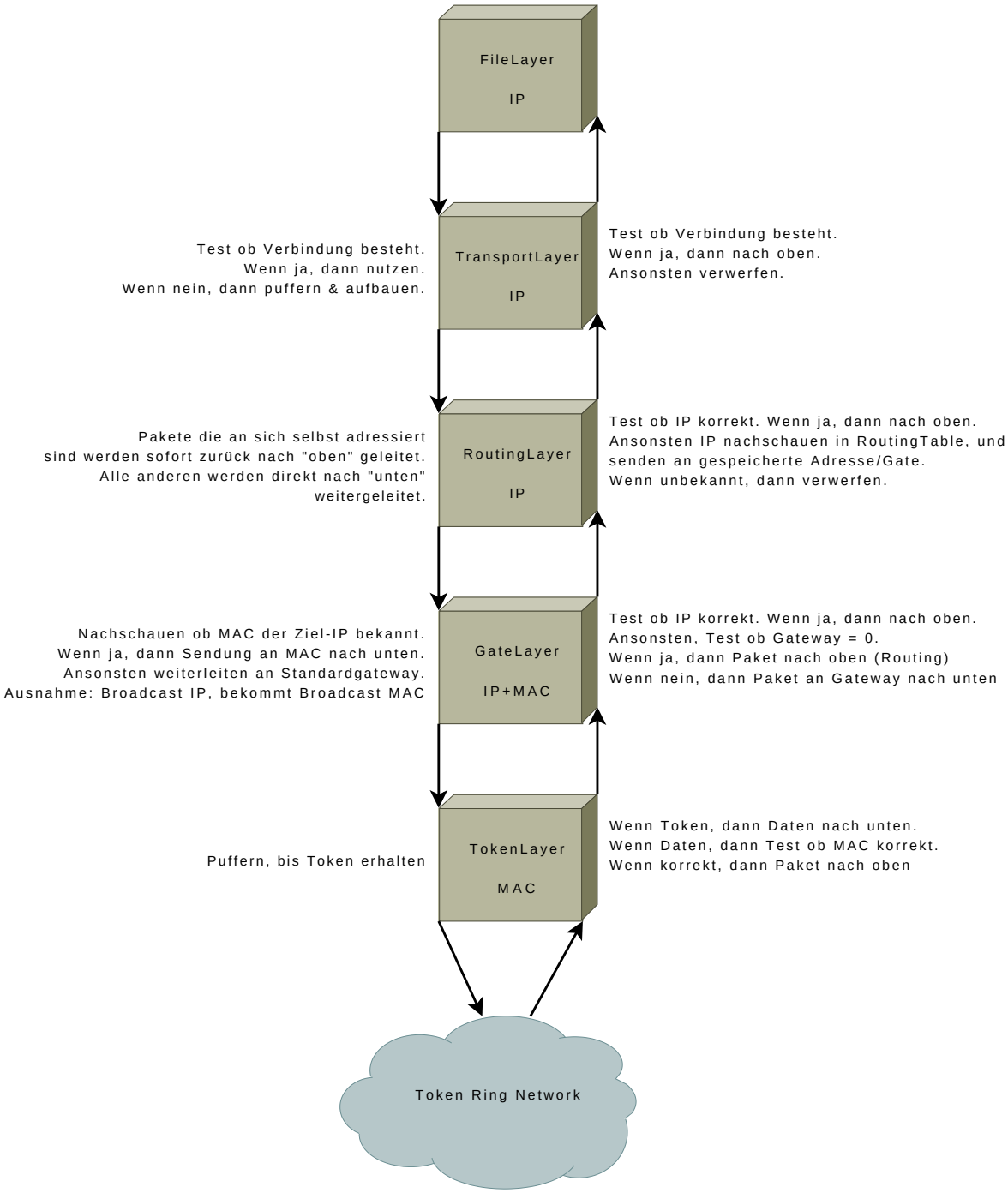

Abbildung 2: Abläufe in einem einfachen PC

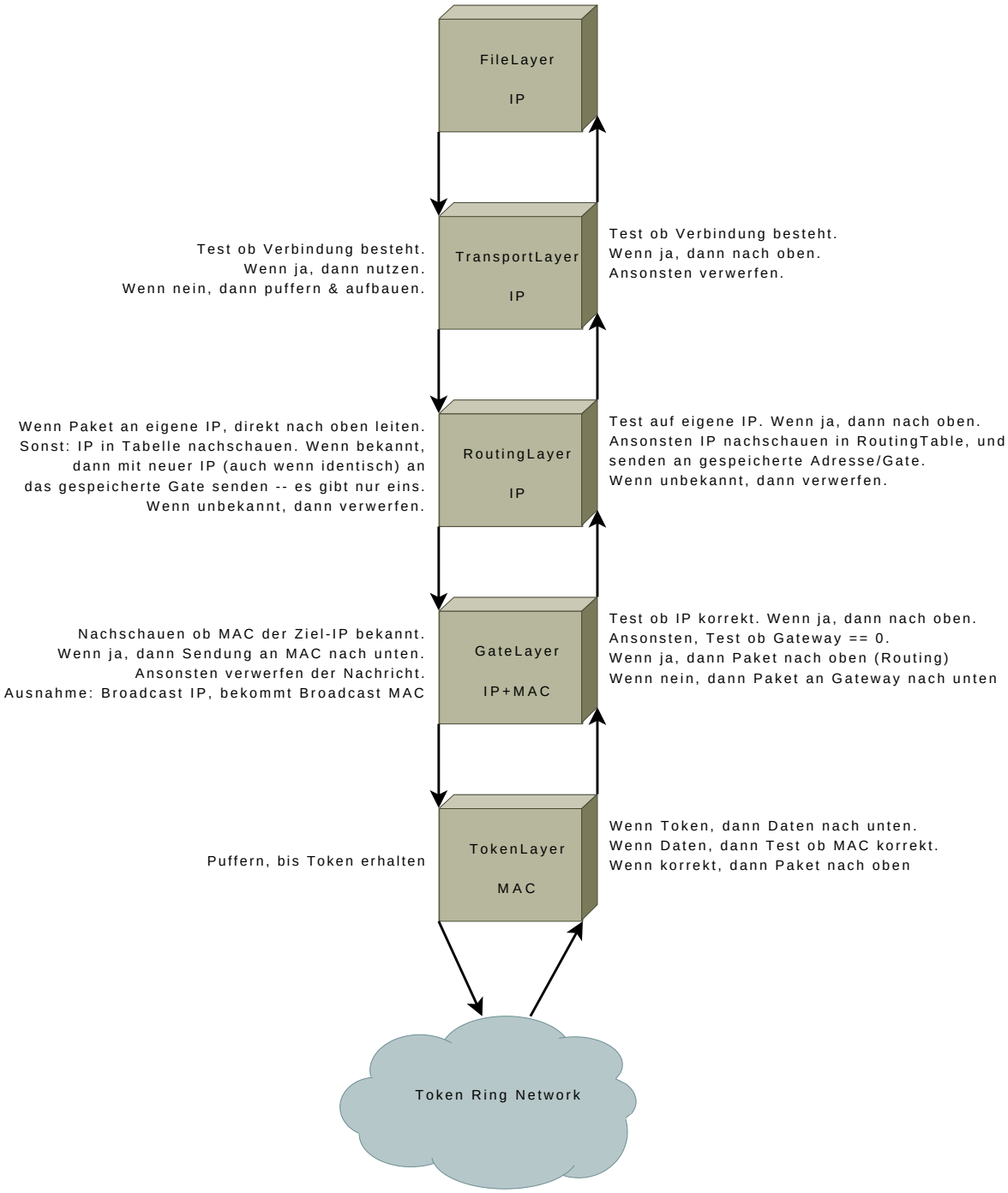

Abbildung 3: Abläufe in einem PC mit aktiver Routing-Schicht

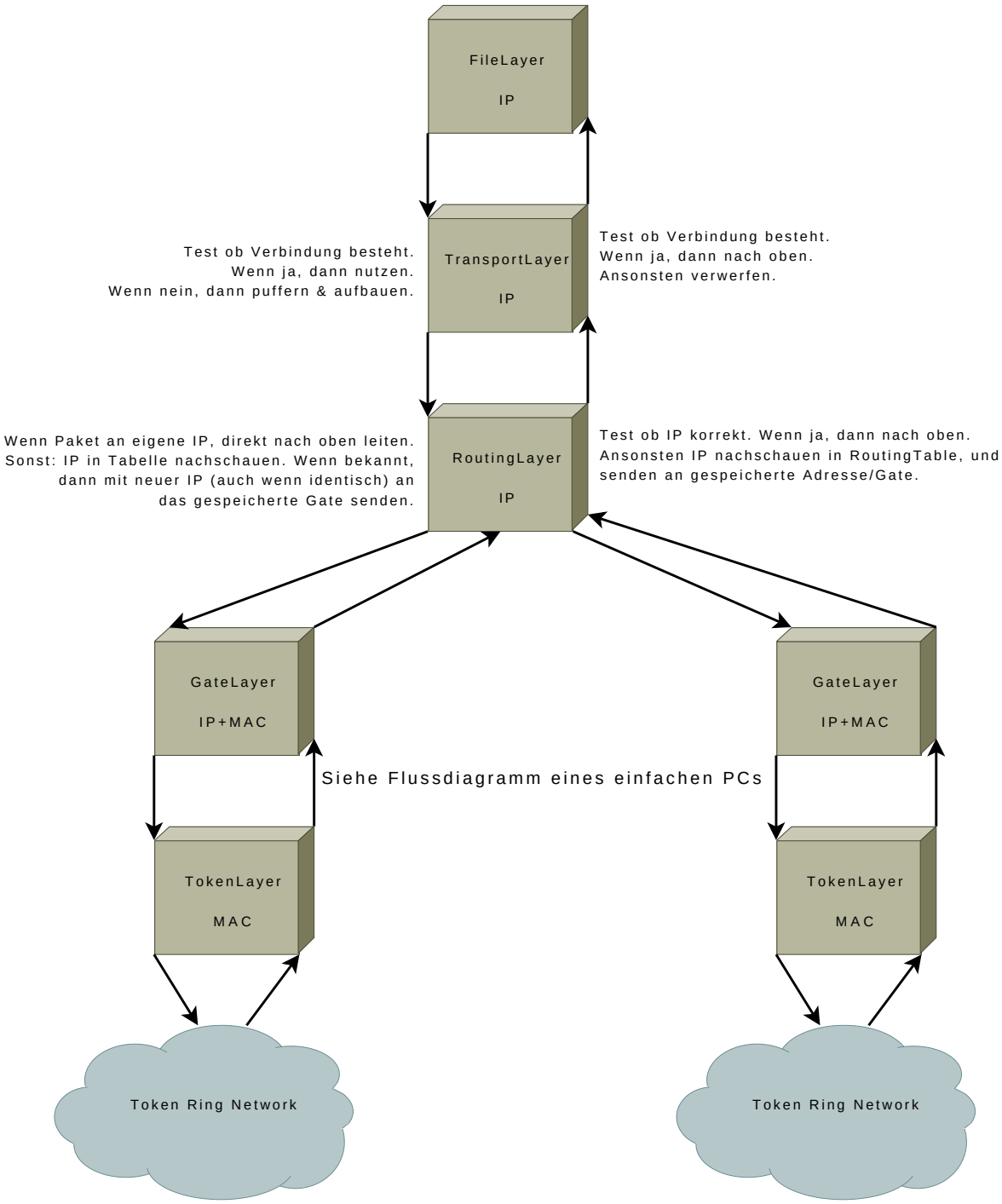

Abbildung 4: Abläufe in einem Router mit mehren Ausgängen

Lizenz