(19) **Bundesrepublik Deutschland**
Deutsches Patent- und Markenamt

(10) **DE 601 17 485 T2** 2006.10.12

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 175 051 B1**
(21) Deutsches Aktenzeichen: **601 17 485.2**
(96) Europäisches Aktenzeichen: **01 306 281.5**
(96) Europäischer Anmeldetag: **20.07.2001**
(97) Erstveröffentlichung durch das EPA: **23.01.2002**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **01.03.2006**
(47) Veröffentlichungstag im Patentblatt: **12.10.2006**

(51) Int Cl.[8]: ***H04L 12/56*** (2006.01)
H04L 29/06 (2006.01)

(30) Unionspriorität:

220026 P	**21.07.2000**	**US**
225630 P	**15.08.2000**	**US**

(73) Patentinhaber:
Hughes Network Systems, LLC, Germantown, Md., US

(74) Vertreter:
Kuhnen & Wacker Patent- und Rechtsanwaltsbüro, 85354 Freising

(84) Benannte Vertragsstaaten:
DE, FR, GB, IT

(72) Erfinder:
Border, John, Poolesville,Maryland 20837, US

(54) Bezeichnung: **Verfahren und Vorrichtung zur Pufferverwaltung**

Beschreibung

HINTERGRUND DER ERFINDUNG

Gebiet der Erfindung

[0001] Die vorliegende Erfindung betrifft ein Kommunikationssystem und betrifft genauer eine Proxy-Architektur zur Verbesserung der Leistungsfähigkeit eines Netzwerks.

Erörterung des technischen Hintergrunds

[0002] Die Einbeziehung von Datennetzwerken in die Abläufe der modernen Gesellschaft, wie durch die Verbreitung des Internet, insbesondere des World Wide Web, augenfällig, stellt immer höhere Anforderungen an Service-Provider, die Leistungsfähigkeit des Netzwerks ständig zu verbessern. Um dieser Herausforderung gewachsen zu sein, investieren Service-Provider massiv in die Aufrüstung ihrer Netzwerke, um die Systemleistung (z.B. die Bandbreite) zu verbessern. Unter vielen Umständen sind solche Aufrüstungen möglicherweise ökonomisch nicht machbar, oder die physischen Beschränkungen des Kommunikationssystems erlauben ein einfaches „Aufrüsten" nicht. Dementsprechend investieren Service-Provider auch in Entwicklungstechnologien, um die Leistungsfähigkeit ihrer Netzwerke zu optimieren. Da viele der heutigen Netzwerke mit der Transmission Control Protocol/Internet Protocol (TCP/IP)-Reihe arbeiten oder diese als Schnittstelle benutzen müssen, konzentrieren sich die Bemühungen auf die Optimierung von TCP/IP-basierten Netzwerkoperationen.

[0003] Als Netzwerkstandard für das globale Internet hat TCP/IP diese Akzeptanz in der Industrie erlangt, weil es flexibel ist und unter Forschern eine lange Tradition hat. Das Transmission Control Protocol (TCP) ist heute das im Internet vorherrschend verwendete Protokoll. TCP basiert auf dem Internet Protocol (IP) und wird in einer Vielfalt von Anwendung, einschließlich der zuverlässigen Datenübertragung und von Internet Web-Seiten-Zugriffsanwendungen, verwendet. Die vier Schichten der TCP/IP-Protokollreihe sind in Fig. 31 erläutert. Wie dargestellt, schließt die Verknüpfungsschicht (oder die Netzschnittstellenschicht) **10** Gerätetreiber im Betriebssystem und etwaige entsprechende Netzschnittstellenkarten ein. Gemeinsam handhaben der Gerätetreiber und die Schnittstellenkarten Hardware-Details der physischen Schnittstellenverbindung mit einem Kabel oder jeder anderen Art von Medium, die verwendet wird. Die Netzwerkschicht (auch als Internetschicht bezeichnet) **12** handhabt die Bewegung von Paketen im Netzwerk. Die Wegewahl bzw. das Routing von Paketen beispielsweise findet in der Netzwerkschicht **12** statt. IP, Internet Control Message Protocol (ICMP) und Internet Group Management Protocol (IGMP) können die Netzwerkschicht in der TCP/IP Protokollreihe bereitstellen. Die Transportschicht **14** sorgt für einen Datenfluss zwischen zwei Hosts für die obige Anwendungsschicht **16**.

[0004] In der TCP/IP Protokollreihe gibt es mindestens zwei verschiedene Transportprotokolle, TCP und ein User Datagram Protocol (UDP). TCP, das für einen zuverlässigen Datenfluss zwischen zwei Hosts sorgt, ist in erster Linie mit dem Aufteilen von Daten, die ihm von der Anwendungsschicht **16** geliefert werden, in Segmente von geeigneter Größe für die darunter liegende Netzwerkschicht **12**, mit der Bestätigung für empfangene Pakete, mit dem Festsetzen von Time-Outs, um sicherzustellen, dass das andere Ende den Empfang der versandten Pakete bestätigt, usw. befasst. Da dieser zuverlässige Datenfluss von der Transportschicht **14** bereitgestellt wird, ist die Anwendungsschicht **16** nicht mit diesen Details befasst. UDP sorgt andererseits für einen wesentlichen einfacheren Dienst für die Anwendungsschicht **16**. UDP sendet einfach Datenpakete, die Datagramme genannte werden, von einem Host zum anderen, ohne Garantie, dass die Datagramme ihren Zielort erreichen. Jede gewünschte Zuverlässigkeit muss von einer höheren Schicht, wie der Anwendungsschicht **16**, hinzugefügt werden.

[0005] Die Anwendungsschicht **16** handhabt die Details der jeweiligen Anwendung. Es gibt viele gemeinsame TCP/IP-Anwendungen, die fast jede Implementierung bereitstellt, einschließlich Telnet für Remote Log-In, das File Transfer Protocol (FTP), das Simple Mail Transfer Protocol (SMTP) oder elektronische Post, das Simple Network Management Protocol (SNMP), das Hypertext Transfer Protocol (HTTP) und viele andere.

[0006] Wie gesagt, stellt TCP eine zuverlässige Datenübermittlung in Reihenfolge zwischen zwei IP-Hosts bereit. Die IP-Hosts stellen eine TCP-Verbindung mit Hilfe eines konventionellen TCP-Dreiwege-Verbindungsaufbaus und eine anschließende Datenübertragung unter Verwendung eines auf Fenstern beruhenden Protokolls mit den erfolgreich empfangenen und bestätigten Daten bereit.

[0007] Um zu verstehen, wo Optimierungen durchgeführt werden können, ist es lehrreich, eine typische TCP-Verbindungseinrichtung zu betrachten. Fig. 32 stellt ein Beispiel für den konventionellen TCP-Dreiwege-Verbindungsaufbau zwischen IP-Hosts **20** und **22** dar. Zuerst schickt der IP-Hosst **20**, der eine Übertragung mit dem IP-Host **22** initiieren will, ein Synchronisierungssignal (SYN) an den IP-Host **22**. Der IP-Host **22** bestätigt das SYN-Signal vom IP-Host **20** durch Senden einer SYN-Bestätigung (ACK). Der dritte Schritt des konventionellen TCP-Dreiwege-Verbindungsaufbaus ist die Ausgabe eines ACK-Signals vom IP-Host **20** an den anderen IP-Host **22**. Nun ist

der IP-Host **22** bereit, die Daten vom IP-Host **20** zu empfangen (und umgekehrt). Nachdem alle Daten versendet wurden, wird ein weiterer Verbindungsaufbau (ähnlich dem Verbindungsaufbau, der beschrieben wurde, um die Verbindung zu initiieren) verwendet, um die TCP-Verbindung zu schließen.

[0008] TCP wurde für große Flexibilität und für die Arbeit über eine große Vielfalt von Kommunikationsverknüpfungen hinweg entworfen, einschließlich von langsamen und schnellen Verbindungen, Verbindungen mit langer Latenz und Verbindungen mit niedrigen und hohen Fehlerraten. Obwohl TCP (und andere Hochschicht-Protokolle) mit vielen verschiedenen Verbindungsarten funktionieren, wird die TCP-Leistung, insbesondere der mögliche Durchsatz durch die TCP-Verbindung, durch die Eigenschaften der verwendeten Verknüpfung beeinflusst. Es müssen viele Verknüpfungsschicht-Designs in Betracht gezogen werden, wenn ein Verknüpfungsschichtdienst entworfen wird, der Internetprotokolle unterstützen soll. Jedoch können nicht alle Eigenschaften durch die Wahl des Verknüpfungsschicht-Designs kompensiert werden. TCP ist dafür entworfen worden, sehr flexibel in Bezug auf die Verknüpfungen zu sein, die es durchläuft. Diese Flexibilität wird auf Kosten eines im Vergleich zu einem maßgeschneiderten Protokoll suboptimalen Betriebs in einer Reihe von Umgebungen erreicht. Dem maßgeschneiderten Protokoll, das in der Regel von proprietärer Natur ist, fehlt die Flexibilität im Hinblick auf Netzwerkumgebungen und Interoperabilität.

[0009] Eine Alternative zu einem maßgeschneiderten Protokoll ist die Verwendung von leistungsverstärkenden Proxies (PEPs), um eine allgemeine Klasse von Funktionen, die „TCP-Spoofing" genannt werden, auszuführen, um die TCP-Leistung gegenüber beeinträchtigen Verknüpfungen (d.h. mit langer Latenz oder hoher Fehlerrate) zu verbessern. TCP-Spoofing schließt eine Zwischen-Netzwerkeinrichtung (den leistungsverstärkenden Proxy (PEP)) ein, die durch Unterbrechen und Abändern, durch die Hinzufügung und/oder Weglassung von TCP-Segmenten das Verhalten der TCP-Verbindung in dem Versuch, deren Leistung zu verbessern, ändert.

[0010] Konventionelle TCP-Spoofing-Implementierungen schließen die lokale Empfangsbestätigung für TCP-Datensegmente ein, damit der Sender der TCP-Daten weitere Daten früher sendet, als es der Fall wäre, wenn kein Spoofing durchgeführt würde, wodurch der Durchsatz der TCP-Verbindung verbessert wird. Im Allgemeinen konzentrieren sich konventionelle TCP-Spoofing-Implementierungen auf die Steigerung des Durchsatzes von TCP-Verbindungen durch entweder die Verwendung größerer Fenster in der Verknüpfung oder durch die Verwendung einer Kompression, um die Datenmenge, die verschickt werden muss, zu verringern, oder beides.

[0011] Viele TCP-PEP-Implementierungen beruhen auf TCP-ACK-Manipulation. Dies kann TCP-ACK-Spacing, wo ACKs, die zusammengepackt sind, voneinander getrennt werden, lokale TCP-ACKs, lokale TCP-Rückübertragungen und TCP-ACK-Filterung und -Rekonstruktion einschließen. Andere PEP-Mechanismen schließen Tunneling, Kompression und auf Priorität beruhendes Multiplexing ein.

[0012] Wie aus den obigen Ausführungen hervorgeht, besteht ein klarer Bedarf an verbesserten Ansätzen für die Optimierung der Netzwerkleistung bei Erreichung einer Netzwerkflexibilität. Es besteht auch die Notwendigkeit, die Leistungsfähigkeit des Netzwerks ohne eine kostspielige Invenstition in die Infrastruktur zu verbessern. Es besteht auch die Notwendigkeit, einen Mechanismus, der die Leistungsfähigkeit eines Netzwerks erhöht, zu verwenden, der mit bestehenden Standards kompatibel ist, um eine schnelle Verbreitung zu erleichtern. Außerdem besteht die Notwendigkeit zur Vereinfachung des Empfänger-Designs. Deshalb ist ein Ansatz zum Optimieren der Leistungsfähigkeit eines Netzwerks mit Hilfe einer Proxy-Architektur sehr wünschenswert.

[0013] EP 0 903 905 A2 beschreibt eine Netzwerk-Gateway-Vorrichtung, die eine Vielzahl von Kommunikationsschnittstellen zum Empfangen von Nachrichten und eine Vielzahl von Modulen, die dafür konfiguriert sind, die Nachrichten zu verarbeiten, um leistungsverbessernde Funktionen zu bewirken, einschließt.

[0014] EP 0 574 140 A1 beschreibt einen Netzwerkadapter, der ein Kopffeld bzw. einen Header und Paketdaten in einem Netzwerkpaket in zwei verschiedene Speichersegmente teilt. Der Header enthält Paddaten, um sicherzustellen, dass die Paketdaten an einer festen Stelle im Netzwerkpaket lokalisiert werden.

Zusammenfassung der Erfindung

[0015] Die vorliegende Erfindung erfüllt die oben genannten Notwendigkeiten durch Bereitstellen einer Netzwerkvorrichtung, die leistungssteigernde Proxy (PEP)-Funktionen bereitstellt. Die Netzwerkvorrichtung schließt mehrere Puffer ein, die Kommunikationsschnittstellen entsprechen und die von leistungssteigernden Proxy (PEP)-Kernels genutzt werden. Die Puffer weisen eine Datenstruktur auf, die ein erweiterbares Feld bereitstellt, das sich an verschiedene Nachrichtenarten anpasst.

[0016] Die verschiedenen Aspekte der vorliegenden Erfindung sind in den beigefügten Ansprüchen definiert.

[0017] Es wird eine Netzwerkvorrichtung zur Bereit-

stellung von Verbesserungen in einem Kommunikationsnetz bereitgestellt. Die Netzwerkvorrichtung schließt eine Vielfalt von Kommunikationsschnittstellen ein, die so konfiguriert sind, dass sie Nachrichten entsprechend einem vorgeschriebenen Protokoll empfangen und weiterleiten. Die Netzwerkvorrichtung schließt auch eine Vielzahl von Modulen ein, die so konfiguriert sind, dass sie die Nachrichten verarbeiten, um leistungssteigernde Funktionen zu bewirken. Ferner schließt die Netzwerkvorrichtung eine Vielzahl von Puffern ein, die so konfiguriert sind, dass sie die empfangenen Nachrichten und Nachrichten, die von einem der vielen Module erzeugt werden, speichern. Ein Teil der Vielzahl von Puffern wird von der Vielzahl von Modulen aufgrund der Durchführung einer der leistungssteigernden Funktionen gemeinsam genutzt. Jeder der Vielzahl von Puffern weist eine Datenstruktur auf, die einen erweiterbaren Header einschließt, um an verschiedene Nachrichtenarten angepasst werden zu können. Dieser Ansatz stellt auf vorteilhafte Weise eine effiziente Pufferverwaltung innerhalb einer Netzwerkkomponente bereit.

[0018] Es wird ein Verfahren zur Bereitstellung von Leistungsverbesserungen für ein Kommunikationsnetz offenbart. Das Verfahren beinhaltet den Empfang von Nachrichten gemäß einem vorgeschriebenen Protokoll, die Verarbeitung der Nachrichten über eine Vielzahl von Modulen, um leistungssteigernde Funktionen zu bewirken, und die Speicherung der empfangen Nachrichten und von Nachrichten, die von einem von der Vielzahl von Modulen erzeugt wurden, in einer Vielzahl von Puffern. Ein Teil der Vielzahl von Puffern wird von der Vielzahl von Modulen aufgrund der Ausführung einer speziellen Leistungsverbesserungsfunktionen gemeinsam genutzt, wobei jeder von der Vielzahl von Puffern eine Datenstruktur hat, die einen erweiterbaren Header einschließt, um sich an verschiedene Nachrichtenarten anpassen zu können. Die oben genannte Anordnung verbessert vorteilhaft die Systemeffizienz.

[0019] Eine Netzwerkvorrichtung zur Bereitstellung von Leistungsverbesserungen für ein Kommunikationsnetz schließt Mittel zum Empfang von Nachrichten entsprechend einem vorgeschriebenen Protokoll und Mittel zum Verarbeiten der Nachrichten, um leistungsverbessernde Funktionen zu bewirken, ein. Die empfangenen Nachrichten und Nachrichten, die von Vearbeitungseinrichtungen erzeugt werden, werden in einer Vielzahl von Puffern gespeichert. Ein Teil der Vielzahl von Puffern wird von den Verarbeitungseinrichtungen aufgrund der Ausführung einer speziellen Leistungsverbesserungsfunktion gemeinsam genutzt. Jeder der Vielzahl von Puffern weist eine Datenstruktur auf, die einen erweiterbaren Header einschließt, um an verschiedene Nachrichtenarten angepasst werden zu können. Die oben genannte Anordnung stellt auf vorteilhafte Weise eine wirksame Pufferverwaltung bereit.

[0020] Ein computerlesbares Medium, das eine oder mehrere Sequenzen aus einem oder mehreren Befehlen zur Bereitstellung von Leistungsverbesserungen eines Kommunikationsnetzes überträgt, wird offenbart. Die eine oder die mehreren Sequenzen aus einem oder mehreren Befehlen schließen Befehle ein, die, wenn sie von einem oder mehreren Rechnern ausgeführt werden, bewirken, dass der eine oder die mehreren Rechner den Schritt des Empfangs von Nachrichten gemäß einem vorgeschriebenen Protokoll ausführen. Andere Schritte beinhalten die Verarbeitung der Nachrichten, um leistungsverbessernde Funktionen zu bewirken, über eine Vielzahl von Modulen und die Speicherung der empfangenen Nachrichten und von Nachrichten, die von einem der Vielzahl von Modulen erzeugt werden, in einer Vielzahl von Puffern. Ein Teil der Vielzahl von Puffern wird von der Vielzahl von Modulen aufgrund der Ausführung einer der leistungssteigernden Funktion gemeinsam genutzt. Jeder der Vielzahl von Puffern weist eine Datenstruktur auf, die einen erweiterbaren Header einschließt, um an verschiedene Nachrichtenarten angepasst werden zu können. Dieser Ansatz stellt auf vorteilhafte Weise eine verbesserte Netzwerkleistung bereit.

[0021] Ein Speicher zum Speichern von Informationen für die Bereitstellung von Leistungsverbesserungen in einem Kommunikationsnetz wird offenbart. Der Speicher weist eine Datenstruktur auf, die ein spezielles Header-Feld einschließt, in dem plattformspezifische Informationen gespeichert werden. Die Datenstruktur schließt auch ein allgemeines Header-Feld ein, in dem Informationen gespeichert werden, die der Vielzahl von Modulen bekannt sind, und ein Nutzinhaltfeld. Mit diesem Ansatz wird eine effiziente Pufferverwaltung erreicht.

Kurze Beschreibung der Zeichnung

[0022] Ein vollständigeres Verständnis der Erfindung und viele der damit zusammenhängenden Vorteile ergibt sich ohne weiteres durch Bezugnahme auf die folgende ausführliche Beschreibung in Zusammenschau mit der begleitenden Zeichnung, worin:

[0023] Fig. 1 ein Diagramm eines Kommunikationssystems ist, in dem der leistungsverbessernde Proxy (PEP) der vorliegenden Erfindung implementiert ist;

[0024] Fig. 2 ein Diagramm einer PEP-Endpunkt-Plattformumgebung gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0025] Fig. 3 ein Diagramm eines TCP-Spoofing-Kernel (TSK) ist, der in der Umgebung von Fig. 2 verwendet wird;

[0026] Fig. 4A und Fig. 4B Ablaufschemata der

Verbindungseinrichtung mit Dreiwege-Verbindungsaufbau-Spoofing bzw. ohne Dreiwege-Verbindungsaufbau-Spoofing sind;

[0027] Fig. 5 ein Diagramm eines PEP-Paketflusses zwischen zwei PEP-Endpunkten gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0028] Fig. 6 ein Diagramm eines IP (Internetprotokoll)-Paketflusses durch einen PEP-Endpunkt gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0029] Fig. 7 ein Diagramm von PEP-Endpunktprofilen, die in der Plattform von Fig. 2 verwendet werden, ist;

[0030] Fig. 8 ein Diagramm der Schnittstellen eines PEP-Endpunkts ist, der als IP-Gateway implementiert ist, gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0031] Fig. 9 ein Diagramm der Schnittstellen eines als Multimedia-Relais implementierten PEP-Endpunkts gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0032] Fig. 10 ein Diagramm der Schnittstellen eines als Multimedia-VSAT (Very Small Aperture Terminal) implementierten PEP-Endpunkts gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0033] Fig. 11 ein Diagramm der Schnittstellen eines in einer Bodenstation implementierten PEP-Endpunkts gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0034] Fig. 12 ein Diagramm des Flusses von TCP-Spoofing-Puffern durch einen erfindungsgemäßen PEP-Endpunkt gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0035] Fig. 13 ein Diagramm der Pufferverwaltung für nicht-gespoofte bzw. nicht-bandbreitenbedarfsreduzierte TCP-Verbindungen und für Nicht-TCP-Verkehr gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0036] Fig. 14 ein Diagramm eines Grundformats der Puffer, die verwendet werden, um die PEP-Funktionalität zu implementieren, gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0037] Fig. 15 ein Diagramm eines IP-Pakets ist, das im System von Fig. 1 verwendet wird;

[0038] Fig. 16 ein Diagramm eines Formats des allgemeinen PEP-Puffer-Headers gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0039] Fig. 17 ein Diagramm einer Header-Anpassung für empfangenen TCP-Datensegmente gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0040] Fig. 18 ein Diagramm eines empfangenen TCP-Datensegments mit einem TCP-Verbindungs-Header gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0041] Fig. 19 ein Diagramm einer Header-Anpassung an eine empfangene TSK-Nachricht gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0042] Fig. 20 ein Diagramm einer Header-Anpassung an eine empfangene TSK-Nachricht mit einem TCP-Verbindungs-Header gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0043] Fig. 21 ein Diagramm eines generierten TCP-Segments gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0044] Fig. 22 ein Diagramm eines generierten PBP-Segments gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0045] Fig. 23 ein Diagramm einer generierten TSK-Nachricht gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0046] Fig. 24 ein Diagramm ist, das die Wiederverwendung eines TCP-Segmentpuffers für eine TSK-Nachricht gemäß einer Ausführungsform der vorliegenden Erfindung zeigt;

[0047] Fig. 25 ein Diagramm der Wiederverwendung eines TSK-Nachrichtenpuffers für ein TCP-Segment gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0048] Fig. 26 ein Diagramm eines Beispiels für eine Kernelverwendung des eigentümer- bzw. inhaberspezifischen „Headers" gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0049] Fig. 27 ein Diagramm eines Verfahrens zum Einsetzen eines allgemeinen PEP-Puffer-Headers in einen kleinen Puffer gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0050] Fig. 28 ein Diagramm eines Verfahrens zum Hinzufügen eines allgemeinen PEP-Puffer-Headers zu einem kleinen Puffer gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0051] Fig. 29 ein Diagramm eines gleitenden Fenstermechanismus, der im System von Fig. 1 verwendet wird, gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0052] Fig. 30 ein Diagramm eines Computersystems, das PEP-Funktionen ausführen kann, gemäß einer Ausführungsform der vorliegenden Erfindung ist;

[0053] Fig. 31 ein Diagramm der Protokollschichten der TCP/IP-Protokollreihe ist; und

[0054] Fig. 32 das Diagramm eines konventionellen TCP-Dreiwege-Verbindungsaufbaus zwischen IP-Hosts ist.

BESCHREIBUNG DER BEVORZUGTEN AUSFÜHRUNGSFORMEN

[0055] In der folgenden Beschreibung sind für den Zweck der Erklärung bestimmte Details ausführlich dargestellt, um ein gründliches Verständnis der Erfindung zu ermöglichen. Jedoch kann die Erfindung natürlich auch ohne diese speziellen Details ausgeführt werden. In einigen Fällen sind bekannte Strukturen und Geräte in Blockdiagrammform dargestellt, um zu vermeiden, dass das Verständnis der Erfindung unnötigerweise erschwert wird.

[0056] Obwohl die vorliegende Erfindung mit Bezug auf das Internet und die TCP/IP-Protokollreihe erörtert wird, kann die vorliegende Erfindung auch auf andere paketvermittelte Netzwerke und äquivalente Protokolle angewendet werden.

[0057] Fig. 1 stellt ein Beispielsnetzwerk **100** dar, in dem der leistungssteigernde Proxy (PEP) der vorliegenden Erfindung verwendet werden kann. Das Netzwerk **100** in Fig. 1 schließt einen oder mehrere Hosts **110** ein, die über TCP-Verbindungen mit einem Netzwerk-Gateway **120** verbunden sind. Das Netzwerk-Gateway **120** ist mit einem anderen Netzwerk-Gateway **140** über eine Rückgrat- bzw. Backbone-Verbindung auf einem Backbone-Link **130** verbunden. Wie in Fig. 1 dargestellt, ist der Backbone-Link **130** in einem Ausführungsbeispiel als Satellitenverknüpfung dargestellt, die über einen Satelliten **101** eingerichtet ist; der Fachmann weiß jedoch, dass auch andere Netzwerkverbindungen implementiert werden können. Zum Beispiel können diese Netzwerkverbindungen im Allgemeinen über ein drahtloses Kommunikationssystem (z.B. Rundfunknetz, Mobilfunknetze usw.) oder ein terrestrisches Kommunikationssystem eingerichtet werden. Das Netzwerk-Gateway **140** ist ferner mit einer zweiten Gruppe von Hosts **150** verbunden, ebenfalls über TCP-Verbindungen. In der in Fig. 1 dargestellten Anordnung erleichtern die Netzwerk-Gateways **120**, **140** die Kommunikation zwischen den Gruppen von Hosts **110**, **150**.

[0058] Die Netzwerk-Gateways **120**, **140**, erleichtern die Kommunikation zwischen den beiden Gruppen von Hosts **110**, **150** durch Ausführen einer Reihe von leistungssteigernden Funktionen. Diese Netzwerk-Gateways **120**, **140** können ein selektives TCP-Spoofing ausführen, was die flexible Konfiguration der speziellen TCP-Verbindungen, deren Bandbreitenbedarf reduziert werden soll, ermöglicht. Außerdem verwenden die Gateways **120**, **140** einen TCP-Dreiwege-Verbindungsaufbau, bei dem die TCP-Verbindungen an jedem Ende des Backbone-Link **130** enden. Von den Netzwerk-Gateways **120**, **140** werden lokale Daten-Empfangsbestätigungen verwendet, wodurch die TCP-Fenster auf lokale Geschwindigkeit beschleunigen können.

[0059] Das Netzwerk-Gateway **120**, **140** multiplext ferner mehrere TCP-Verbindungen über eine einzelne Backbone-Verbindung; diese Fähigkeit reduziert das Maß an Empfangsbestätigungsverkehr im Zusammenhang mit den Daten von mehreren TCP-Verbindungen, da eine einzige Backbone-Verbindungs-Empfangsbestätigung verwendet werden kann. Die Multiplexfunktion unterstützt außerdem TCP-Verbindungen mit hohem Durchsatz, in denen das Backbone-Verbindungsprotokoll für den speziellen verwendeten Backbone-Link optimiert ist. Die Netzwerk-Gateways **120**, **140** unterstützen außerdem die Datenkompression über dem Backbone-Link **130**, um das Maß an zu sendendem Verkehr zu reduzieren, was die Leistung der Backbone-Verbindung weiter verstärkt. Weiter verwenden die Netzwerk-Gateways **120**, **140** eine Datenverschlüsselung bei der Datenübertragung über den Backbone-Link **130**, um die Datensicherheit zu gewährleisten, und stellen die Fähigkeit zum priorisierten Zugang zum Backbone-Link **130** auf Basis einer TCP-Verbindung bereit. Jeder der Netzwerk-Gateways **120**, **140** kann einen besonderen Pfad wählen, auf dem die Daten, die mit einer Verbindung assoziiert sind, fließen. Die oben genannten Fähigkeiten der Netzwerk-Gateways **120**, **140** sind nachstehend ausführlicher beschrieben.

[0060] Fig. 2 stellt einen leistungssteigernden Proxy (PEP) **200** dar, dar in einem Netzwerk-Gateway **120**, **140** entsprechend einer Ausführungsform der vorliegenden Erfindung implementiert ist. In dieser Ausführungsform weist der PEP **200** eine Plattformumgebung **210** auf, welche das Hardware- und Software-Betriebssystem einschließt. Der PEP **200** schließt auch lokale Netzwerk (LAN)-Schnittstellen **220** und Weitverkehrsnetzwerk (WAN)-Schnittstellen **230** ein. Im Beispiel von Fig. 1 kann der Netzwerk-Gateway **120** die TCP-Verbindungen mit den IP-Hosts **110** über eine lokale LAN-Schnittstelle **220** einrichten und kann die Backbone-Verbindung mit dem Netzwerk-Gateway **140** über eine WAN-Schnittstelle **230** einrichten. Die PEP-Plattformumgebung **210** kann auch allgemeine funktionelle Module einschließen: ein Routingmodul **240**, ein Pufferverwaltungsmodul **250**, ein Ereignisverwaltungsmodul **260** und ein Parameterverwaltungsmodul **270**. Wie in

Fig. 2 dargestellt, schließt der Netzwerk-Gateway auch einen TCP-Spoofing-Kernel (TSK) **280**, einen Backbone-Protokoll-Kernel (BPK) **282**, einen Priorisierungs-Kernel (PK) **284** und einen Pfadauswahl-Kernel (PSK) **286** ein. Diese vier Kernels machen im Wesentlichen die Funktionalität des leistungsverbessernden Proxy **200** aus.

[0061] Die Plattformumgebung **210** führt eine Anzahl von Funktionen aus. Eine dieser Funktion ist die Abschirmung der verschiedenen PEP-Kernels **280**, **282**, **284** gegen die Implementierung spezifischer Beschränkungen. Das heißt, die Plattformumgebung **210** führt Funktionen aus, die die verschiedenen PEP-Kernels **280**, **282**, **284**, **286** nicht direkt ausführen können, weil die Implementierung der Funktion plattformspezifisch ist.

[0062] Diese Anordnung hat den Vorteil, plattformspezifische Details vor den PEP-Kernels **280**, **282**, **284**, **286** zu verbergen, wodurch die PEP-Kernels portierbarer werden. Ein Beispiel für eine plattformspezifische Funktion ist die Zuweisung eines Puffers. In einigen Plattformen werden Puffer nach Bedarf geschaffen, während in anderen Plattformen Puffer zu Beginn geschaffen werden und für die spätere Verwendung in verknüpften Listen organisiert werden. Es sei darauf hingewiesen, dass plattformspezifische Funktionen nicht auf Funktionen beschränkt sind, die für alle Kernels **280**, **282**, **284**, **286** auswählbar sind. Es kann auch eine Funktion, die für einen bestimmten Kernel spezifisch ist, beispielsweise die Zuweisung eines Steuerblocks für das TCP-Spoofing, von der Plattformumgebung implementiert werden, um plattformspezifische Details vor dem Kernel zu verbergen.

[0063] Außerdem kann die Plattformumgebung **210** den Aufgabenkontext liefern, in dem die PEP-Kernels **280**, **282**, **284**, **286** arbeiten. In einem Ausführungsbeispiel können alle PEP-Kernels **280**, **282**, **284**, **286** aus Gründen der Effizienz in demselben Aufgabenkontext arbeiten. Jedoch ist dies nicht erforderlich.

[0064] Ferner liefert die Plattformumgebung **210** eine Schnittstelle zwischen der PEP-Funktion (ausgeführt in den Kernels **280**, **282**, **284**, **286**) und den anderen Funktionen des Netzwerk-Gateways **120**, **140**. Zum Beispiel kann die Plattformumgebung **210** die Schnittstelle zwischen der PEP-Funktionalität und der Routing-Funktion **240** bereitstellen, wie in Fig. 2 dargestellt. Es sei darauf hingewiesen, dass die in Fig. 2 dargestellten plattformspezifischen Funktionen Beispiele sind und nicht als vollständige Liste anzusehen sind. Es sei ferner darauf hingewiesen, dass die dargestellten PEP-Kernels, die einander in Fig. 2 berühren (**280**, **282** und **284**, **286**), eine direkte prozedurale Schnittstelle miteinander haben können. Weiter können die Kernels **280**, **282**, **284**, **286** direkte Schnittstellen einschließen, um die Leistung zu verbessern, anstatt alles durch die Plattformumgebung **210** zu führen (wie in Fig. 2 dargestellt).

[0065] Zusätzlich zu den PEP-Kernels **280**, **282**, **284** und **286** kann die PEP-Endpunkt-Plattform **210** einen Datenkompressions-Kernel (CK) **290** und einen Verschlüsselungs-Kernel (EK) **292** verwenden. Wie oben beschrieben, erleichtern diese Kernels **280**, **282**, **284**, **286**, **290** und **292** die Kommunikation zwischen den zwei Gruppen von Hosts **110**, **150** durch die Ausführung einer Vielfalt von leistungsverbessernden Funktionen entweder einzeln oder in Kombination. Diese leistungsverbessernden Funktionen schließen selektives TCP-Spoofing, Dreiwege-Verbindungsaufbau-Spoofing, lokale Datenempfangsbestätigung, Multiplexing von TCP-Verbindung zu Backbone-Verbindung, Datenkompression/-verschlüsselung, Priorisierung und Pfadauswahl ein.

[0066] Das selektive TCP-Spoofing wird vom TSK **280** ausgeführt und schließt einen Satz von durch den Anwender konfigurierbaren Regeln ein, die verwendet werden, um zu bestimmen, welche TCP-Verbindungen bandbreitenbedarfsreduziert werden sollen. Das selektive TCP-Spoofing verbessert die Leistung durch Nicht-Einbeziehen von mit TCP-Spoofing in Beziehung stehenden Ressourcen, wie dem Pufferplatz, Steuerblöcken usw., für TCP-Verbindungen, für die der Anwender bestimmt hat, dass ein Spoofing nicht von Vorteil oder nicht erforderlich ist, und durch Unterstützen der Verwendung von maßgeschneiderten Parametern für bandbreitenbedarfsreduzierte TCP-Verbindungen.

[0067] Genauer unterscheidet der TSK **280** die verschiedenen TCP-Verbindungen aufgrund der Anwendungen, welche diese verwenden. Das heißt, TSK **280** unterscheidet diese TCP-Verbindungen, um zu bestimmen, welche Verbindung bandbreitenbedarfsreduziert werden soll, ebenso wie die Art, auf die die Verbindung bandbreitenbedarfsreduziert wird; z.B. ob der Bandbreitenbedarf des Dreiwege-Verbindungsaufbaus reduziert wird, die speziellen Zeitüberschreitungsparameter für die bandbreitenbedarfsreduzierten Verbindungen usw. Dann wird das TCP-Spoofing nur für jene TCP-Verbindungen ausgeführt, die Anwendungen zugeordnet sind, für die ein hoher Durchsatz oder eine reduzierte Anfangslatenz der Verbindung (oder beides) erforderlich ist. Infolgedessen hebt sich der TSK **280** TCP-Spoofing-Ressourcen nur für die TCP-Verbindungen auf, für die ein hoher Durchsatz oder eine reduzierte Anfangslatenz der Verbindung (oder beides) erforderlich ist. Ferner erhöht der TSK **280** die Gesamtzahl von TCP-Verbindungen, die aktiv sein können, bevor die TCP-Spoofing-Ressourcen ausgehen, da aktive TCP-Verbindungen, die keinen hohen Durchsatz erfordern, nicht zugeordnete Ressourcen sind.

[0068] Ein Kriterium zum Identifizieren von

TCP-Verbindungen von Anwendungen, für die ein TCP-Spoofing ausgeführt werden sollte und nicht ausgeführt werden sollte, ist das in den verschickten TCP-Paketen enthaltene TCP-Port-Nummernfeld. Im Allgemeinen sind jeder Art von Anwendung eindeutige Port-Nummern zugeteilt. Welche TCP-Port-Nummern bandbreitenbedarfsreduziert werden sollen und welche nicht, kann im TSK **280** gespeichert werden. Der TSK **280** ist auch umkonfigurierbar, um einem Anwender oder einem Operator zu ermöglichen, die TCP-Port-Nummern, die bandbreitenbedarfsreduziert und die nicht bandbreitenbedarfsreduziert werden sollen, neu zu konfigurieren. Der TSK **280** erlaubt es einem Anwender oder einem Operator außerdem, aufgrund von anderen Kriterien zu steuern, welche TCP-Verbindungen bandbreitenbedarfsreduziert werden sollen. Im Allgemeinen kann eine Entscheidung darüber, ob eine TCP-Verbindung bandbreitenbedarfsreduziert werden soll, auf jedem Feld innerhalb des TCP-Pakets beruhen. Der TSK **280** erlaubt es einem Anwender, anzugeben, welche Felder zu prüfen sind und welche Werte in diesen Feldern TCP-Verbindungen identifizieren, die bandbreitenbedarfsreduziert werden sollen oder die nicht bandbreitenbedarfsreduziert werden sollen. Ein weiteres Beispiel für einen potentiellen Nutzen dieser Fähigkeit ist, dass der Anwender oder Operator die IP-Adresse des TCP-Pakets wählen kann, um zu steuern, für welche Anwender das TCP-Spoofing ausgeführt wird. Der TSK **280** ermöglicht es einem Anwender außerdem, mehrere Felder gleichzeitig zu betrachten. Infolgedessen ermöglicht es der TSK 280 einem Anwender oder Operator, mehrere Kriterien zu verwenden, um die TCP-Verbindungen, deren Bandbreitenbedarf reduziert werden soll, zu wählen. Zum Beispiel kann der Systemoperator durch Wählen sowohl der IP-Adresse als auch der TCP-Port-Nummernfelder ein TCP-Spoofing nur für bestimmte Anwendungen von bestimmten Anwendern ermöglichen.

[0069] Die vom Anwender konfigurierbaren Regeln können fünf Beispielskriterien einschließen, die vom Anwender oder Operator festgelegt werden können, um eine selektive TCP-Spoofing-Regel zu erzeugen: Ziel-IP-Adresse; Quell-IP-Adresse; TCP-Portnummern (die sowohl die TCP-Ziel- aus auch die -Quell-Portnummern betreffen können); TCP-Optionen und nach IP unterschiedenes Dienst (DS)-Feld. Wie oben angegeben, können jedoch auch andere Felder innerhalb des TCP-Pakets verwendet werden.

[0070] Wie oben erörtert, können zusätzlich zur Unterstützung von selektiven TCP-Spoofing-Regeln für jedes dieser Kriterien UND- und ODER-Kombinationsoperatoren verwendet werden, um Kriterien miteinander zu verknüpfen. Zum Beispiel kann durch Verwenden des UND-Kombinationsoperators eine Regel definiert werden, um das TCP-Spoofing für FTP-Daten, die von einem bestimmten Host erhalten werden, nicht zuzulassen. Auch kann die Reihenfolge, in der die Regeln festgelegt sind, wichtig sein. Es ist möglich, dass eine Verbindung den Kriterien mehrerer Regeln genügt. Deshalb kann der TSK **280** Regeln in der Reihenfolge anwenden, die vom Operator festgelegt wurde, wobei er die Aktion der ersten Regel nimmt, die passt. Es kann auch eine Voreinstellungs- bzw. Default-Regel festgesetzt werden, welche die Aktion definiert, die für TCP-Verbindungen ergriffen wird, die zu keiner der definierten Regeln passen. Der vom Operator ausgewählte Regelsatz kann in einem selektiven TCP-Spoofing-Auswahlprofil definiert werden.

[0071] Angenommen, es wurde ausreichend Pufferspeicherplatz zugewiesen, um beispielsweise den Bandbreitenbedarf von fünf TCP-Verbindungen zu reduzieren, und wenn vier langsame Anwendungen (d.h. Anwendungen, die von Natur aus keine hohe Geschwindigkeit erfordern) Verbindungen zusammen mit einer schnellen Anwendung errichten, dann hat die schnelle Anwendung Zugriff auf nur 1/5 des verfügbaren Pufferspeicherplatzes. Falls fünf langsame Verbindungen vor der schnellen Verbindung zustande kommen, kann der Bandbreitenbedarf der schnellen Verbindung gar nicht reduziert werden. Unter Verwendung des selektiven TSK 280-Spoofingmechanismus wird den langsamen Verbindungen keinerlei Spoofing-Pufferplatz zugewiesen. Daher hat die schnelle Verbindung immer Zugriff auf den Pufferplatz, was ihre Leistung im Vergleich zu einer Implementierung ohne das selektive TCP-Spoofing-Merkmal des TSK **280** verbessert.

[0072] Der TSK **280** erleichtert auch das Spoofing des konventionellen Dreiwege-Verbindungsaufbaus. Ein Dreiwege-Verbindungsaufbau-Spoofing beinhaltet lokale Antworten auf eine Verbindungsanfrage, um eine TCP-Verbindung zustande zu bringen, parallel zur Sendung der Verbindungsanfragen über den Backbone-Link **130** (Fig. 1). Dadurch wird es dem Ursprungs-IP-Host (beispielsweise **110**) möglich, den Punkt zu erreichen, wo er in der Lage ist, die Daten, die er verschicken muss, mit lokalen Geschwindigkeiten zu verschicken, d.h. Geschwindigkeiten, die unabhängig von der Latenz des Backbone-Links **130** sind. Das Dreiwege-Verbindungsaufbau-Spoofing macht es möglich, die Daten, die der IP-Host **110** verschicken muss, zum Ziel-Host **150** zu schicken, ohne auf die durchgehende bzw. Ende-zu-Ende-Einrichtung der TCP-Verbindung zu warten. Für Backbone-Links mit hoher Latenz reduziert dies erheblich die Zeit, die nötig ist, um die TCP-Verbindung zustande zu bringen, und was am wichtigsten ist, die Zeit, die es insgesamt dauert, um eine Antwort (von einem IP-Host **150**) auf die Daten zu erhalten, die der IP-Host **110** schickt.

[0073] Ein spezifisches Beispiel, in dem diese Technik von Nutzen ist, betrifft eine Internet-Web-Seiten-Zugriffsanwendung. Beim Dreiwege-Verbin-

dungsaufbau-Spoofing kann die Anfrage eines IP-Host nach Zugriff auf eine Web-Seite unterwegs zum Web-Server sein, ohne auf die Ende-zu-Ende-Einrichtung der TCP-Verbindung warten zu müssen, wodurch die Zeit, die der Download der Web-Seite benötigt, verkürzt wird.

[0074] Mit Local Data Acknowledgement bestätigt der TSK **280** im Netzwerk-Gateway **120** (beispielsweise) lokal den Empfang von Datensegmenten, die vom IP-Host **110** erhalten wurden. Dadurch kann der IP-Host **110** sofort weitere Daten verschicken. Noch wichtiger ist, dass TCP eingegangene Empfangsbestätigungen als Signale zur Erhöhung der aktuellen TCP-Fenstergröße nutzt. Infolgedessen ermöglicht die lokale Empfangsbestätigung es dem versendenden IP-Host **110**, das TCP-Fenster wesentlich rascher zu vergrößern als dies durch Ende-zu-Ende-Empfangsbestätigungen unterstützt wird. Der TSK **280** (der Spoofer) übernimmt Verantwortung für die zuverlässige Zustellung der von ihm bestätigten Daten.

[0075] Im BPK **282** werden mehrere TCP-Verbindungen auf einer einzigen Backbone-Verbindung gemultiplext und übertragen. Dadurch wird die Systemleistung verbessert, da die Daten für mehrere TCP-Verbindungen von einer einzigen Backbone-Verbindungs-Empfangsbestätigung (ACK) bestätigt werden können, was die Menge des Empfangsbestätigungsverkehrs, der erforderlich ist, um einen hohen Durchsatz durch den Backbone-Link **130** aufrechtzuerhalten, erheblich reduziert. Darüber hinaus wählt der BPK **282** ein Backbone-Verbindungsprotokoll aus, das dafür optimiert ist, einen hohen Durchsatz für die spezielle Verknüpfung bereitzustellen. Unterschiedliche Backbone-Verbindungsprotokolle können vom BPK **282** mit verschiedenen Backbone-Links verwendet werden, ohne die grundlegende TCP-Spoofing-Implementierung zu ändern. Das vom BPK **282** ausgewählte Backbone-Verbindungsprotokoll stellt eine ausreichende Unterstützung für die zuverlässige, schnelle Sendung von Daten über den Backbone-Link **130** bereit, wobei die Details der Behinderungen (beispielsweise hohe Latenz) der Verknüpfung gegenüber der TCP-Spoofing-Implementierung verborgen bleiben.

[0076] Das Multiplexen durch den BPK **282** ermöglicht die Verwendung eines Backbone-Link-Protokolls, das individuell auf die Verwendung mit der speziellen Verknüpfung zugeschnitten ist und eine Technik zur Verstärkung der Leistung des Backbone-Link-Protokolls mit wesentlich weniger Abhängigkeit von der Einzelleistung der bandbreitenbedarfsreduzierten TCP-Verbindungen bereitstellt als herkömmliche Methoden. Ferner macht die Fähigkeit, das Backbone-Protokoll für verschiedene Backbone-Links maßzuschneidern, die vorliegende Erfindung auf unterschiedliche Systeme anwendbar.

[0077] Der PEP **200** kann optional einen Datenkompressions-Kernel **290** zum Komprimieren von TCP-Daten und einen Verschlüsselungs-Kernel **292** zum Verschlüsseln von TCP-Daten einschließen. Die Datenkompression erhöht die Datenmenge, die über die Backbone-Verbindung übertragen werden kann. Verschiedene Kompressionsalgorithmen können vom Datenkompressions-Kernel **290** unterstützt werden, und mehr als eine Kompressionsart kann gleichzeitig unterstützt werden. Der Datenkompressions-Kernel **290** kann optional eine Kompression auf TCP-Verbindungsbasis anwenden, bevor die TCP-Daten mehrerer TCP-Verbindungen auf der Backbone-Verbindung oder aufgrund einer Backbone-Verbindung gemultiplext werden, nachdem die TCP-Daten verschiedener TCP-Verbindungen auf der Backbone-Verbindung gemultiplext wurden. Welche Option genutzt wird, wird dynamisch nach vom Anwender konfigurierten Regeln und den spezifischen genutzten Kompressionsalgorithmen bestimmt. Beispiele für Datenkompressionsalgorithmen sind in den US-Patenten Nr. 5,973,630, 5,955,976 offenbart, deren gesamter Inhalt durch Bezugnahme hierin aufgenommen ist. Der Verschlüsselungs-Kernel verschlüsselt die TCP-Daten für die sichere Übertragung über den Backbone-Link **130**. Die Verschlüsselung kann anhand jeder herkömmlichen Technik durchgeführt werden. Es ist auch klar, dass der entsprechende Spoofer (in dem oben ausgeführten Beispiel der Network-Gateway **140**) geeignete Kernel für die Dekompression und Verschlüsselung einschließt, die beide anhand von beliebigen herkömmlichen Techniken durchgeführt werden können.

[0078] Der PK **284** stellt einen priorisierten Zugang zur Backbone-Link-Leistung bereit. Beispielsweise kann die Backbone-Verbindung tatsächlich in N (N > 1) unterschiedliche Unterverbindungen geteilt werden, von denen jede eine andere Prioritätshöhe aufweist. In einem Ausführungsbeispiel können vier Prioritätshöhen unterstützt werden. Der PK **284** nutzt anwenderdefinierte Regeln, um verschiedenen TCP-Verbindungen verschiedene Prioritäten und somit unterschiedliche Unterverbindungen der Backbone-Verbindung zuzuweisen. Es sei darauf hingewiesen, dass PK **284** auch Nicht-TCP-Verkehr (z.B. UDP (User Datagram Protocol)-Verkehr) priorisieren kann, bevor er den Verkehr über den Backbone-Link **130** schickt.

[0079] Der PK **284** nutzt auch anwenderdefinierte Regeln zur Steuerung der Menge der Leistung des Backbone-Link **130**, die jeder Prioritätshöhe zur Verfügung steht. Beispiele für Kriterien, die verwendet werden können, um die Priorität zu bestimmen, schließen die folgenden ein: Ziel-IP-Adresse; Quell-IP-Adresse; nächstes IP-Protokoll; TCP-Portnummern (was sowohl die TCP-Ziel- als auch -Quell-Portnummern betreffen kann); UDP-Portnummern (was sowohl die UDP-Ziel- als auch die

-Quell-Portnummern betreffen kann); und IP-DiffServ (DS)-Feld. Die Art der Daten in den TCP-Datenpaketen kann ebenfalls als Kriterium genommen werden. Beispielsweise könnten Videodaten die höchste Priorität erhalten. Zielkritische Daten könnten ebenfalls die höchste Priorität erhalten. Wie beim selektiven TCP-Spoofing kann jedes Feld im IP-Paket vom PK **284** verwendet werden, um die Priorität zu bestimmen. Es sei jedoch darauf hingewiesen, dass unter solchen Szenarios die Folge der Verwendung eines solchen Felds bewirken könnte, das verschiedene IP-Pakete des gleichen Flusses (z.B. TCP-Verbindung) unterschiedliche Prioriäten erhalten können; diese Szenarios sollten vermieden werden.

[0080] Wie oben angegeben, können zusätzlich zur Unterstützung selektiver Priorisierungsregeln für jedes dieser Kriterien UND- und ODER-Kombinationsoperatoren verwendet werden, um Kriterien miteinander zu verknüpfen. Zum Beispiel kann unter Verwendung des UND-Kombinationsoperators eine Regel definiert werden, um eine Priorität für SNMP-Daten, die von einem bestimmten Host empfangen werden, zuzuordnen. Auch die Reihenfolge, in der die Befehle festgelegt werden, kann von Bedeutung sein. Es ist möglich, dass eine Verbindung den Kriterien mehrerer Regeln entspricht. Daher kann der PK **284** Regel in der vom Operator festgelegten Reihenfolge anwenden, wobei er die erste Regel ausführt, die passt. Es kann auch eine Default-Regel festgesetzt werden, die definiert, welche Aktionen für IP-Pakete ausgeführt werden sollen, die keiner der definierten Regeln entsprechen. Der vom Operator ausgewählte Regelsatz kann in einem Priorisierungsprofil definiert werden.

[0081] Was die Pfadauswahlfunktionalität betrifft, so ist der PSK **286** verantwortlich für die Bestimmung, welchen Weg ein IP-Paket einschlagen soll, um sein Ziel zu erreichen. Der vom PSK **286** ausgewählte Weg kann durch Anwenden von Wegauswahlregeln bestimmt werden. Der PSK **286** bestimmt auch, welche IP-Pakete unter Verwendung eines alternativen Wegs versendet werden sollen und welche IP-Pakete fallen gelassen werden sollen, wenn einer oder mehrere primäre Wege ausfallen. Pfadauswahlparameter können auch anhand von Profilen konfiguriert werden. Die Pfadauswahlregeln können so ausgelegt sein, dass sie Flexibilität im Hinblick auf die Zuordnung von Pfaden verleihen, während sie sicherstellen, dass alle Pakete, die zum gleichen Verkehrsfluss (z.B. der gleichen TCP-Verbindung) gehören, den gleichen Pfad nehmen (obwohl es auch möglich ist, Segmente der gleichen TCP-Verbindung auf unterschiedlichen Wegen zu verschicken, könnte dieses Segment-„Splitting" negative Nebenwirkungen haben). Beispiele für Kriterien, die verwendet werden können, um einen Pfad auszuwählen, schließen die folgenden ein: Priorität des IP-Pakets wie vom PK **284** eingestellt (sollte das üblichste Kriterium sein), Ziel-IP-Adresse; Quell-IP-Adresse; nächstes IP-Protokoll; TCP-Portnummern (was sich sowohl auf die TCP-Ziel- als auch die -Quell-Portnummern beziehen kann); UDP-Portnummern (was sich sowohl auf die UDP-Ziel- als auch die -Quell-Portnummern beziehen kann); und IP-DiffServ (DS)-Feld. Ähnlich wie beim selektiven TCP-Spoofing und Priorisierung kann der PSK **284** einen Pfad unter Verwendung jedes beliebigen Felds im IP-Paket bestimmen.

[0082] Was die Priorisierungskriterien (Regeln) betrifft, so können die UND- und ODER-Kombinationsoperatoren verwendet werden, um Kriterien miteinander zu verknüpfen. Beispielsweise kann anhand des UND-Kombinationsoperators eine Regel definiert werden, um einen Pfad für SNMP-Daten auszuwählen, die von einem bestimmten Host empfangen werden. Auch kann die Reihenfolge wichtig sein, in der die Regeln festgelegt sind. Es ist möglich, dass eine Verbindung den Kriterien mehrerer Regeln entspricht. Daher kann der PSK **286** Regeln in der vom Operator festgelegten Reihenfolge anwenden, wobei er die erste passende Regel ausführt. Es kann auch eine Default-Regel festgesetzt werden, die definiert, welche Aktionen für IP-Paketen durchgeführt werden sollen, die keiner der definierten Regeln entsprechen. Der vom Operator ausgewählte Regelsatz kann in einem Pfadauswahlprofil definiert sein.

[0083] Beispielsweise kann eine Pfadauswahlregel den Pfad aufgrund der folgenden Pfadinformationen wählen, in denen die IP-Pakete die folgende Regel erfüllen: in einem primären Pfad, einem sekundären Pfad, einem tertiären Pfad. Der primäre Pfad wird in jeder Pfadauswahlregel festgelegt. Der sekundäre Pfad wird nur benutzt, wenn der primäre Pfad ausgefallen ist. Falls kein sekundärer Pfad festgelegt wurde, können beliebige IP-Pakete, die der Regel entsprechen, fallen gelassen werden, wenn der primäre Pfad ausfällt. Der tertiäre Pfad wird nur festgelegt, wenn ein sekundärer Pfad festgelegt wurde. Der tertiäre Pfad wird ausgewählt, wenn sowohl der primäre Pfad als auch der sekundäre Pfad ausgefallen sind. Falls kein tertiärer Pfad festgelegt wurde, können beliebige IP-Pakete, die der Regel entsprechen, fallen gelassen werden, wenn sowohl der primäre als auch der sekundäre Pfad ausfallen. Die Pfadauswahl kann verallgemeinert werden, so dass die Pfadauswahlregel bis zu N Pfade auswählen kann, wobei der N. Pfad nur ausgewählt wird, wenn der (N – 1). Pfad ausfällt. Das obige Beispiel, wo N = 3, dient nur als Beispiel, obwohl N in der Regel eine ziemlich kleine Zahl ist.

[0084] Anhand eines Beispiels wird die Betriebsweise des Systems **100** wie folgt beschrieben. Zunächst wird eine Backbone-Verbindung zwischen den PEPs von zwei Netzwerk-Gateways **120**, **140** (d.h. den beiden Spoofern), die sich an den jeweiligen Enden des Backbone-Link **130**, für den ein TCP-Spoofing ge-

wünscht ist, befinden, eingerichtet. Sobald ein IP-Host **110** eine TCP-Verbindung initiiert, überprüft der TSK **280** des PEP **200**, der am IP-Host **110** vor Ort ist, seine konfigurierten TCP-Spoofing-Regeln. Falls die Regeln anzeigen, dass der Bandbreitenbedarf der Verbindung nicht reduziert werden soll, erlaubt der PEP **200** den durchgehend nicht-bandbreitenbedarfsreduzierten Fluss der Verbindung. Falls die Regeln anzeigen, dass der Bandbreitenbedarf der Verbindung reduziert werden soll, antwortet der bandbreitenbedarfsreduzierende PEP **200** lokal auf den TCP-Dreiwege-Verbindungsaufbau des IP-Hosts. Parallel dazu schickt der PEP **200** eine Nachricht über den Backbone-Link **130** zu seinem Partner-Netzwerk-Gateway **140** und bittet diesen, einen TCP-Dreiwege-Verbindungsaufbau mit dem IP-Host **150** an dessen Seite des Backbone-Link **130** zu initiieren. Dann werden Daten zwischen dem IP-Host **110**, **150** mit dem PEP **200** des Network-Gateway **120** ausgetauscht, der die empfangenen Daten lokal bestätigt und diese über den Backbone-Link **130** durch die schnelle Backbone-Verbindung schickt, wobei er die Daten komprimiert wie aufgrund der konfigurierten Komprimierungsregeln angemessen. Die Priorität der TCP-Verbindung wird bestimmt, wenn die Verbindung eingerichtet ist. Der BPK **282** kann die Verbindung mit anderen empfangenen Verbindungen über eine einzige Backbone-Verbindung multiplexen, der PK **284** bestimmt die Priorität der Verbindung und der PSK **286** bestimmt den Pfad, den die Verbindung einschlagen soll.

[0085] Wie oben beschrieben, verbessert der PEP **200** auf vorteilhafte Weise die Netzwerkleistung durch Zuordnen von mit TCP-Spoofing in Beziehung stehenden Ressourcen, wie Pufferspeicherplatz, Steuerblöcken usw., nur zu TCP-Verbindungen, für die ein Spoofing Vorteile bringt; durch Spoofing des Dreiwege-Verbindungsaufbaus zum Verkürzen der Datenantwortzeit; durch Reduzieren der Zahl von ACKs, die durch Ausführen einer lokalen Empfangsbestätigung übertragen werden, und durch Bestätigen von mehreren TCP-Verbindungen mit einem einzigen ACK; durch Durchführen einer Datenkomprimierung, um die Datenmenge, die übertragen werden kann, zu erhöhen; durch Zuordnen von Prioritäten zu verschiedenen Verbindungen und durch Definieren von mehreren Pfaden für einzurichtende Verbindungen.

[0086] Fig. 3 zeigt ein Beispiel für einen Stapel, der die Beziehung zwischen dem TCP-Stapel und den PEP-Kernels **280**, **282**, **284**, **286** der vorliegenden Erfindung zeigt. Der TSK **280** ist in erster Linie verantwortlich für Funktionen im Zusammenhang mit TCP-Spoofing. Der TSK **280** schließt in einem Ausführungsbeispiel zwei Grundelemente ein: eine Transportschicht, die einen TCP-Stapel **303** und einen IP-Stapel **305** umfasst; und eine TCP-Spoofing-Anwendung **301**. Die Transportschicht ist verantwortlich für die Interaktion innerhalb der TCP-Stapels (z.B. **303**) der IP-Hosts **110**, die mit einer lokalen LAN-Schnittstelle **220** eines PEP **210** verbunden sind.

[0087] Der TSK **280** implementiert das TCP-Protokoll, das die geeigneten TCP-Statusmaschinen einschließt und beendet bandbreitenbedarfsreduzierte TCP-Verbindungen. Die TCP-Spoofing-Anwendung **301** sitzt auf der Transportschicht auf und dient als Anwendung, die Daten von Anwendungen der IP-Hosts **110** empfängt und an diese verschickt. Aufgrund der Schichtarchitektur des Protokolls isoliert die TCP-Spoofing-Anwendung **301** die Details des TCP-Spoofing gegenüber der Transportschicht, wodurch die Transportschicht auf Standardweise arbeiten kann.

[0088] Wie in Fig. 3 dargestellt, kann die TCP-Spoofing-Anwendung **301** auch eine Verknüpfung mit der BPK **282** eingehen, die mit den WAN-Schnittstellen **230** in Verbindung steht. Der BPK sorgt für die Aufrechterhaltung des Backbone-Protokolls, wobei er die das Protokoll, mit dem die Netzwerk-Gateways **120**, **140** (in Fig. 1) kommunizieren, implementiert. Der BPK **282** sorgt für eine zuverlässige Sendung von Daten, nutzt einen relativ kleinen Umfang an Bestätigungsverkehr und unterstützt die Verwendung von ausgewählten Backbones (d.h. eine Verwendung, die nicht spezifisch für die TSK **280** ist); eines dieser Beispiele ist das Reliable Data Protocol (RDP).

[0089] Der BPK **282** liegt gemäß einem Ausführungsbeispiel über dem PK **284** und dem PSK **286**. Der PK **284** ist dafür verantwortlich, die Priorität von IP-Paketen zu bestimmen und dann aufgrund der Priorität Übertragungsmöglichkeiten zuzuordnen. Der PK **284** kann auch den Zugriff auf Pufferspeicherpuffer steuern, indem er die Warteschlangengröße, die mit dem Verschicken und Empfangen von Paketen assoziiert ist, steuert. Der PSK **286** bestimmt, welchen Weg ein IP-Paket nehmen soll, um sein Ziel zu erreichen. Der Pfad, der vom PSK **286** ausgewählt wird, kann durch Anwenden von Pfadauswahlregeln bestimmt werden. Der PSK **286** kann auch bestimmen, welches IP-Paket unter Verwendung eines alternativen Pfads weitergegeben werden soll und welche Pakete fallen gelassen werden sollen, wenn einer oder mehrere primäre Pfade ausfallen.

[0090] Die Fig. 4A und Fig. 4B zeigen Ablaufschemata der Einrichtung einer bandbreitenbedarfsreduzierten TCP-Verbindung unter Verwendung von Dreiwege-Verbindungsaufbau-Spoofing bzw. ohne Dreiwege-Verbindungsaufbau-Spoofing. Der TCP-Spoofing-Kernel **280** richtet eine bandbreitenbedarfsreduzierte TCP-Verbindung ein, wenn ein TCP <SYN>-Segment von seinem lokalen LAN oder eine

Connection Request-Nachricht von seinem TSK-Peer empfangen wird. Es sei darauf hingewiesen, dass das Dreiwege-Verbindungsaufbau-Spoofing außer Funktion gesetzt werden kann, um einen Austausch mit maximaler Ende-zu-Ende-Segmentgröße (MSS) zu unterstützen, wie nachstehend ausführlicher beschrieben. Für den Zweck der Erläuterung wird das Verfahren zum Einrichten einer bandbreitenbedarfsreduzierten TCP-Verbindung mit Bezug auf einen lokalen Host **400**, einen lokalen PEP-Endpunkt **402**, einen entfernten PEP-Endpunkt **404** und einen entfernten Host **406** beschrieben. Wie bereits erwähnt, stellt der TSK **280** innerhalb der einzelnen PEP-Endpunkte **402** und **404** die Spoofing-Funktion bereit.

[0091] In Schritt **401** überträgt der lokale Host **400** ein TCP <SYN>-Segment an den lokalen PEP-Endpunkt **402** an einer lokalen LAN-Schnittstelle **220**. Wenn ein TCP-Segment von der lokalen LAN-Schnittstelle **220** empfangen wird, bestimmt die Plattformumgebung **402**, ob bereits ein TCP-Verbindungs-Steuerblock (CCB) der zu dem TCP-Segment gehörenden TCP-Verbindung zugeordnet ist. Falls kein CCB vorhanden ist, überprüft die Umgebung **402**, ob das TCP-Segment ein <SYN>-Segment ist, das an ein nicht-lokales Ziel geschickt wird. Falls dies der Fall ist, stellt das <SYN>-Segment einen Versuch dar, eine neue (nicht lokale) TCP-Verbindung zustande zu bringen, und die Umgebung **402** gibt das Segment an den TCP-Spoofing-Kernel **280** ab, um die Disposition der TCP-Verbindung zu bestimmen. Wenn ein TCP <SYN>-Segment von der lokalen LAN-Schnittstelle **220** für eine neue TCP-Verbindung empfangen wird, bestimmt der TCP-Spoofing-Kernel **280** zuerst, ob der Bandbreitenbedarf der Verbindung reduziert werden soll. Falls der Bandbreitenbedarf der Verbindung reduziert werden soll, nutzt der TSK **280** (in einem Ausführungsbeispiel) die in dem ausgewählten TCP-Spoofing-Parameterprofil angezeigte Priorität und den Peer-Index (der von der Umgebung **210** mit dem TCP <SYN>-Segment bereitgestellt wird), um den Handle der Backbone-Verbindung zu konstruieren, der verwendet werden soll, um diese bandbreitenbedarfsreduzierte TCP-Verbindung zu befördern. In dem Ausführungsbeispiel wird der Peer-Index als die 14 Bits höherer Ordnung des Handle verwendet, und die Priorität wird als die beiden Bits niederer Ordnung des Handle verwendet. Der Backbone-Verbindungs-Handle wird dann (über die TSK-Steuerblock (TCB)-Mappingtabelle) verwendet, um den TCB zu finden, der mit der Backbone-Verbindung assoziiert ist. Der TSK **280** des PEP-Endpunkts **402** überprüft dann, ob die Backbone-Verbindung aktiv ist. Falls die Backbone-Verbindung aktiv ist, bestimmt der TSK **280** ob, die Zahl der bandbreitenbedarfsreduzierten TCP-Verbindungen, die bereits die ausgewählte Backbone-Verbindung nutzen, immer noch unter dem CCB-Ressourcen-Limit liegt. Das CCB-Ressourcen-Limit ist die kleinere von der lokalen Zahl der CCBs (bereitgestellt als Parameter durch die Plattformumgebung **210**) und der Peer-Zahl der CCBs (empfangen in der letzten TSK-Peerparameter (TPP)-Nachricht vom TSK-Peer), die für diese Backbone-Verbindung zur Verfügung steht. Falls die Zahl der Verbindungen noch immer unter dem Limit liegt, ordnet der TSK **280** des PEP-Endpunkts **402** der Verbindung einen einzigartigen TCP-Verbindungsidentifizierer (z.B. einen freien CCB-Mappingtabellen-Eintragsindex) zu und ruft die Umgebung **210**, um einen TCP-Verbindungssteuerblock für die Verbindung zuzuordnen.

[0092] Der TSK **280** des PEP-Endpunkts **402** schicht das TCP <SYN>-Segment zurück zur Umgebung **210**, um nicht-bandbreitenbedarfsreduziert weitergeschickt zu werden, falls die obigen Prüfungen versagen. Anders ausgedrückt führen die folgenden Bedingungen dazu, dass die TCP-Verbindung nicht bandbreitenbedarfsreduziert ist. Zuerst, wenn die selektiven TCP-Spoofing-Regeln anzeigen, dass die Verbindung nicht bandbreitenbedarfsreduziert werden soll. Auch besteht keine Backbone-Verbindung für die Priorität, bei der der Bandbreitenbedarf der TCP-Verbindung reduziert werden sollte (angezeigt durch die Anwesenheit eines TCB für die Backbone-Verbindung). Kein Spoofing wird durchgeführt, wenn die Backbone-Verbindung nicht aktiv ist. Außerdem wird, wenn die Zahl der bandbreitenbedarfsreduzierten TCP-Verbindungen, die bereits die Backbone-Verbindung nutzen, einen vorgegebenen Schwellenwert erreicht oder übertrifft, dann kein Spoofing durchgeführt. Falls kein CCB-Mapping-Tabelleneintrag verfügbar ist oder wenn kein CCB aus dem freien CCB-Pool verfügbar ist, dann wird die TCP-Verbindung ohne Reduzierung des Bandbreitenbedarfs weitergegeben. In dem Fall, dass keine Backbone-Verbindung besteht, kann der TSK **280** des PEP-Endpunkts **402** auch ein Ereignis posten, um den Operator zu warnen, dass ein Missverhältnis zwischen den konfigurierten TCP-Spoofing-Parameterprofilen und dem konfigurierten Satz an Backbone-Verbindungen besteht.

[0093] Wenn man das Beispiel fortführt, so schreibt der TSK **280** des PEP-Endpunkts **402**, falls alle obigen Prüfungen bestanden werden, den Backbone-Verbindungs-Handle in den Pufferspeicher, in dem das TCP <SYN>Segment hinterlegt ist. Es sei darauf hingewiesen, dass dies nicht geschieht, solange kein CCB erfolgreich von der Plattformumgebung **402** zugeordnet wurde, da diese Umgebung die Pufferspeicher nicht zählt, bis ein CCB erfolgreich zugeordnet wurde. Der TSK **280** kopiert dann die Parameter von dem ausgewählten TCP-Spoofing-Parameterprofil in den CCB. Als Folge davon werden relevante Informationen (z.B. die maximale Segmentgröße, die von dem Host geboten wird (falls kleiner als die konfigurierte MSS), die Anfangssequenznummer usw.) aus dem TCP <SYN>-Segment kopiert

und in dem CBB gespeichert. Es sei darauf hingewiesen, dass die Quell- und Ziel-IP-Adressen und Quell- und Ziel-TCP-Portnummern von der Plattformumgebung **402** bereits in den CCB eingegeben wurden, als der CCB zugeordnet wurde; die Umgebung **402** nutzt diese Informationen, um CCB-Hash-Funktionkollisionen zu handhaben.

[0094] Nach Zuordnen und Festsetzen des CCB konstruiert der TCP-Spoofing-Kernel des PEP-Endpunkts **402** eine Connection Request (CR)-Nachricht in Schritt **403** und schickt diese zu seinem TSK-Peer, der mit dem entfernten PEP-Endpunkt **404** assoziiert ist. Die CR-Nachricht enthält im Grunde alle Informationen, die aus dem TCP-Spoofing-Parameterprofil und dem TCP <SYN>-Segment extrahiert wurden und die im lokalen CCB gespeichert wurden, z.B. die Quell- und Ziel-IP-Adressen, die Quell- und Ziel-TCP-Portnummern, den MSS-Wert usw., mit der Ausnahme von Feldern, die nur lokale Bedeutung haben, wie die Anfangssequenznummer. (Die IP-Adressen und TCP-Portzahlen werden in einen TCP-Verbindungs-Header eingegeben.) Anders ausgedrückt, die CR-Nachricht enthält alle Informationen, die der Peer-TSK des PEP-Endpunkts **404** braucht, um seinen eigenen CCB einzurichten. Um die Einrichtung der lokalen Verbindung zu beenden, schickt der TCP-Spoofing-Kernel **280** des lokalen PEP-Endpunkts **402** in Schritt **405** ein TCP <SYN,ACK>-Segment an den lokalen Host **400** als Antwort auf das erhaltene <SYN>-Segment. Der TSK **280** des PEP-Endpunkts **402** führt Schritt **405** gleichzeitig mit dem Schritt des Versendens der Connection Request-Nachricht (d.h. Schritt **403**) durch, falls ein Dreiwege-Verbindungsaufbau-Spoofing möglich ist. Ansonsten wartet der TSK **280** von **402** auf eine Connection Established (CE)-Nachricht von seinem TSK-Peer vom entfernten PEP-Endpunkt **404**, bevor er das <SYN,ACK>-Segment versendet. In einem Ausführungsbeispiel wählt der TSK **280** des PEP-Endpunkts **402** eine zufällige Anfangssequenznummer (wie in der IETF (Internet Engineering Task Force) RFC **793** bereitgestellt, die hierin in ihrer Gesamtheit durch Bezugnahme aufgenommen ist), die zum Versenden von Daten verwendet wird.

[0095] Falls das Dreiwege-Verbindungsaufbau-Spoofing nicht möglich ist, ist der MSS-Wert, der im <SYN,ACK>-Segment verschickt wird, gleich dem MSS-Wert, der in der CE-Nachricht erhalten wird. Falls ein Dreiwege-Verbindungsaufbau-Spoofing möglich ist, wird der MSS-Wert aus dem TCP-Spoofing-Parameterprofil bestimmt, das für die Verbindung (und die konfigurierte maximale Pfadübertragungseinheit (MTU)) gewählt wurde. Für diesen Fall vergleicht der TSK **280** des PEP-Endpunkts **402** dann den MSS-Wert, der in der Connection Established-Nachricht erhalten wird, wenn er ankommt, mit dem Wert, der zum lokalen Host im TCP <SYN,ACK>-Segment geschickt wird. Falls der MSS-Wert, der in der CE-Nachricht erreicht wird, kleiner ist als der MSS-Wert, der zum lokalen Host geschickt wird, liegt ein Missverhältnis der maximalen Segmentgröße vor. (Falls ein MSS-Missverhältnis vorliegt, kann es notwendig sein, dass der TSK die Größe von TCP-Datensegmenten anpasst, bevor er sie versendet.) Nach Versenden des TCP <SYN,ACK>-Segments (Schritt **405**) ist der TSK **280** des lokalen PEP-Endpunkts **402** bereit, Daten von dem lokalen Host **400** zu empfangen. In Schritt **407** überträgt der lokale Host **400** ein <ACK>-Segment an den TSK **280** des PEP-Endpunkts **402**; danach schickt der lokale Host, wie in Schritt **409**, auch Daten an den TSK **280** des PEP-Endpunkts **402**. Wenn ein Dreiwege-Verbindungsaufbau-Spoofing angewendet wird, muss der TSK **280** nicht darauf warten, dass die Connection Established-Nachricht von seinem TSK-Peer ankommt, bevor er Daten empfängt und weiterschickt. Wie in **Fig. 4A** dargestellt, schickt der TSK **280** des lokalen PEP-Endpunkts **402** in Schritt **411** ein <ACK>-Segment an den lokalen Host und sendet gleichzeitig die TCP-Daten (TD) von dem lokalen Host **400** zum Peer-TSK des PEP-Endpunkts **404** (in Schritt **413**), bevor er eine CE-Nachricht vom TSK des PEP-Endpunkts **404** empfängt.

[0096] Der TSK **280** des PEP-Endpunkts **402** nimmt jedoch keine Daten von seinem TSK-Peer vom PEP-Endpunkt **404** an, bis die CE-Nachricht empfangen wurde. Der TSK **280** des PEP-Endpunkts **402** schickt keine Daten, die er von seinem TSK-Peer des PEP-Endpunkts **404** empfangen hat, zum lokalen Host **400** weiter, bis er das TCP <ACK>-Segment empfangen hat, das anzeigt, dass der lokale Host das <SYN,ACK>-Segment empfangen hat (wie in Schritt **407**).

[0097] Wenn eine Connection Request-Nachricht von einem Peer-TSK empfangen wird (Schritt **403**), ordnet der TCP-Spoofing-Kernel **280** einen CCB für die Verbindung zu und speichert dann alle relevanten Informationen aus der CR-Nachricht im CCB. Der TSK **280** des PEP-Endpunkts **404** nutzt dann diese Informationen, um ein TCP <SYN>-Segment zu erzeugen, wie in Schritt **415**, um es zum entfernten Host **406** zu schicken. Die MSS im <SYN>-Segment wird auf den Wert eingestellt, der vom TSK-Peer des PEP-Endpunkts **404** erhalten wurde. Wenn der entfernte Host mit einem TCP <SYN,ACK>-Segment antwortet (Schritt **417**) schickt der TSK **280** des PEP-Endpunkts **402** eine Connection Established-Nachricht an seinen TSK-Peer des entfernten PEP-Endpunkts **404** (Schritt **419**), wobei er in der CE-Nachricht die MSS, die von dem lokalen Host in dem <SYN,ACK>-Segment geschickt wird, einschließt. Der TSK **280** des PEP-Endpunkts **402** antworte ebenfalls, wie in Schritt **421**, mit einem TCP <ACK>-Segment, um den lokalen Dreiwege-Verbindungsaufbau zu vervollständigen. Der Peer-TSK des PEP-Endpunkts **404** schickt dann in Schritt **423** die

Daten, die er vom TSK **280** empfangen hat, zum Host weiter. Gleichzeitig schickt der entfernte Host **406** in Schritt **425** Daten an den Peer-TSK vom PEP-Endpunkt **404**, der in Schritt **427** den Empfang der Daten durch Ausgabe eines <ACK>-Segments an den entfernten PEP-Endpunkt **404** bestätigt. Gleichzeitig mit der Bestätigung werden die Daten an den TSK **280** des PEP-Endpunkts **402** geschickt (Schritt **429**).

[0098] An diesem Punkt ist der TSK **280** bereit, Daten aus jeder Richtung zu empfangen und weiterzusenden. Der TSK **280** schickt die Daten, wie in Schritt **431**, an den lokalen Host weiter, der seinerseits ein <ACK>-Segment verschickt (Schritt **433**). Falls die Daten von ihrem TSK-Peer ankommen, bevor eine <SYN,ACK>-Segment-Antwort vom lokalen Host erhalten wird, werden die Daten in die Warteschlange gestellt und dann verschickt, nachdem das <ACK>-Segment als Antwort auf ein <SYN,ACK>-Segment verschickt wird (wenn dieses ankommt).

[0099] Wie in Fig. 4B dargestellt, wird eine bandbreitenbedarfsreduzierte TCP-Verbindung eingerichtet, wenn ein Dreiwege-Verbindungsaufbau-Spoofing nicht möglich ist. In diesem Szenario übermittelt der lokale Host **400** ein TCP <SYN>-Segment, wie in Schritt **451**, an den TSK **280** im lokalen PEP-Endpunkt **402**. Anders als bei der TCP-Verbindungseinrichtung von Fig. 4A antwortet der lokale PEP-Endpunkt **402** auf das TCP <SYN>-Segment nicht mit einem <SYN,ACK>-Segment, sondern schickt einfach eine CR-Nachricht an den entfernten PEP-Endpunkt **404** (Schritt **453**). Als Nächstes wird in Schritt **455** ein TCP <SYN>-Segment an den entfernten Host **406** geschickt. Als Antwort darauf übermittelt der entfernte Host **406** ein TCP <SYN,ACK>-Segment zurück an den entfernten PEP-Endpunkt **404** (in Schritt **457**). Danach schickt der entfernte PEP-Endpunkt **404**, wie in Schritt **459**, eine CE-Nachricht an den lokalen PEP-Endpunkt **402**, der anschließend in Schritt **461** ein <SYN,ACK>-Segment an den lokalen Host **400** ausgibt. Gleichzeitig mit Schritt **459** gibt der entfernte PEP-Endpunkt **404** ein <ACK>-Segment an den entfernten Host **406** aus (Schritt **463**).

[0100] Sobald er das <ACK>-Segment empfängt, kann der entfernte Host **406** in Schritt **465** mit der Datenübertragung beginnen. Sobald der PEP-Endpunkt **404** die Daten vom entfernten Host **406** empfängt, übermittelt der entfernte PEP-Endpunkt **404** gleichzeitig, wie in Schritt **467**, die TD-Nachricht an den lokalen PEP-Endpunkt **402** und übermittelt ein <ACK>-Segment an den entfernten Host **406**, um den Datenempfang zu bestätigen (Schritt **469**).

[0101] Da der lokale Host **400** ein <SYN,ACK>-Segment vom lokalen PEP-Endpunkt **402** empfangen hat, bestätigt der lokale Host **400** die Nachricht in Schritt **471**. Danach übermittelt der lokale Host **400** Daten an den lokalen PEP-Endpunkt **402**. In diesem Beispiel schickt der lokale PEP-Endpunkt **402**, bevor er die Daten vom lokalen Host **400** empfängt, die Daten, die vom entfernten Host **406** stammen, in Schritt **475** über die TD-Nachricht (Schritt **467**) an den lokalen Host **400**.

[0102] Als Antwort auf die empfangenen Daten (in Schritt **473**) gibt der lokale PEP-Endpunkt **402** ein <ACK>-Segment aus, wie in Schritt **477**, und schickt die Daten in Schritt **479** in einer TD-Nachricht an den entfernten PEP-Endpunkt **404**. Der lokale Host **400** antwortet auf die in Schritt **475** empfangenen Daten mit einem <ACK>-Segment an den lokalen PEP-Endpunkt **402** (Schritt **481**). Der entfernte PEP-Endpunkt **404** schickt die Daten vom lokalen Host **400**, wie in Schritt **483**, nachdem er die TD-Nachricht empfangen hat. Nach Empfang der Daten bestätigt der entfernte Host **406** den Empfang durch Zurücksenden eines <ACK>-Segments an den entfernten PEP-Endpunkt **404** in Schritt **485**.

[0103] Fig. 5 zeigt den Fluss von Paketen mit der PEP-Architektur gemäß einer Ausführungsform der vorliegenden Erfindung. Wie dargestellt, schließt ein Kommunikationssystem **500** einen Hub-Site- (oder lokalen) PEP-Endpunkt **501** ein, der eine Konnektivität mit einem Fern-Site-PEP-Endpunkt **503** über eine Backbone-Verbindung hat. Beispielsweise handhaben an der Hub-Site (oder der lokalen Site) und an jeder entfernten Site PEP-Endpunkte **501** und **503** IP-Pakete. Der PEP-Endpunkt **501** schließt ein Internal IP-Paket-Routing-Modul **501a** ein, das lokale IP-Pakete empfängt und diese Pakete mit einem TSK **501b** und einem BPK **501c** tauscht. Ebenso schließt der entfernte PEP-Endpunkt **503** ein internes IP-Paket-Routing-Modul **503a** ein, das mit einem TSK **503b** und einem BPK **503c** kommuniziert. Abgesehen von der Tatsache, dass der Hub-Site PEP-Endpunkt **501** mehr Backbone-Protokoll-Verbindungen unterstützen kann als ein Fern-Site-PEP-Endpunkt **503**, ist die Hub- und die Fern-Site-PEP-Verarbeitung symmetrisch.

[0104] Für den Lokal-zu-WAN-Verkehr (d.h. in Aufwärtsrichtung), empfängt der PEP-Endpunkt **501** IP-Pakete von seiner lokalen Schnittstelle **220** (Fig. 2). Nicht-TCP-IP-Pakete werden an die WAN-Schnittstelle **230** weitergeschickt (nach Bedarf) (Fig. 2). TCP-IP-Pakete werden intern an den TSK **501b** weitergeschickt. TCP-Segmente, die zu Verbindungen gehören, die nicht bandbreitenbedarfsreduziert werden, werden vom Spoofing-Kernel **501b** zum Routing-Modul **501a** zurückgeschickt, um unmodifiziert an die WAN-Schnittstelle **230** weitergeschickt zu werden. Für bandbreitenbedarfsreduzierte TCP-Verbindungen beendet der TCP-Spoofing-Kernel **501a** die TCP-Verbindung lokal. TCP-Daten, die von einer bandbreitenbedarfsreduzierten Verbindung empfangen werden, werden vom Spoofing-Kernel **501a** zum

Backbone-Protokoll-Kernel **501c** weitergegeben und dann auf der geeigneten Backbone-Protokollverbindung gemultiplext. Der Backbone-Protokoll-Kernel **501c** stellt sicher, dass die Daten über das WAN zugestellt werden.

[0105] Für WAN-zu-Lokal-Verkehr (d.h. in Abwärtsrichtung) empfängt der entfernte PEP-Endpunkt **503** IP-Pakete von seiner WAN-Schnittstelle **230** (Fig. 2). IP-Pakete, die nicht an den Endpunkt **503** adressiert sind, werden einfach (nach Bedarf) an die lokale Schnittstelle **220** (Fig. 2) weitergeschickt. IP-Pakete, die an den Endpunkt **503** adressiert sind und die einen nächsten Protokoll-Header-Typ „PBP" aufweisen, werden zum Backbone-Protokoll-Kernel **503c** weitergeschickt. Der Backbone-Protokoll-Kernel **503c** extrahiert die TCP-Daten und schickt diese an den TCP-Spoofing-Kernel **503b** für die Übertragung an die geeignete bandbreitenbedarfsreduzierte TCP-Verbindung weiter. Zusätzlich zum Transport der TCP-Daten wird die Backbone-Protokoll-Verbindung vom TCP-Spoofing-Kernel **501b** verwendet, um Steuerinformationen zu seinem Peer-TCP-Spoofing-Kernel **503b** im entfernten PEP-Endpunkt **503** zu schicken, um Verbindungseinrichtungen und Verbindungsbeendigungen zu koordinieren.

[0106] Eine Priorisierung kann an vier Punkten im System **500** innerhalb des Routings **501a** und TSK **501b** vom PEP-Endpunkt **501** und innerhalb des Routing **503a** und TSK **503b** vom PEP-Endpunkt **503** vorgenommen werden. In Aufwärtsrichtung werden Prioritätsregeln an die Pakete einzelner TCP-Verbindungen am Zugangspunkt zum TCP-Spoofing-Kernel **501b** angewendet. Diese Regeln erlauben es dem Anwender, zu steuern, welche Anwendungen höheren oder niedrigen Prioritätszugang zu Spoofing-Ressourcen haben. Eine aufwärtsgerichtete Polarisierung wird ebenfalls durchgeführt, bevor die Pakete an das WAN weitergeschickt werden. Dies ermöglicht es dem Kunden, die relative Priorität von bandbreitenbedarfsreduzierten TCP-Verbindungen im Vergleich zu nicht-bandbreitenbedarfsreduzierten TCP-Verbindungen und Nicht-TCP-Verkehr zu steuern (ebenso wie die Steuerung der relativen Priorität dieser anderen Verkehrsarten in Beziehung zueinander). Auf der abwärtsgerichteren Seite wird die Priorisierung verwendet, um den Zugriff auf Pufferspeicherplatz und andere Ressourcen im PEP-Endpunkt **503** allgemein und im Hinblick auf TCP-Spoofing zu steuern.

[0107] An der Hub- (oder der lokalen) Site kann der PEP-Endpunkt **501** in einem Netzwerk-Gateway (z.B. einem IP-Gateway) gemäß einer Ausführungsform der vorliegenden Erfindung implementiert sein. An der entfernten Site kann der PEP-Endpunkt **503** in der Fern-Site-Komponente, z.B. einem Satellitentermial wie einem Multimedia-Relais, einem Multimedia-VSAT oder einem Personal Earth Station (PES) Remote, implementiert sein.

[0108] Die Architektur des Systems **500** stellt eine Reihe von Vorteilen bereit. Zunächst kann das TCP-Spoofing sowohl in Aufwärts- als auch in Abwärtsrichtung durchgeführt werden. Außerdem unterstützt das System das Spoofing des TCP-Verbindungsstarts und ein selektives TCP-Spoofing, bei dem nur der Bandbreitenbedarf von Verbindungen, die von einem Spoofing profitieren, tatsächlich reduziert wird. Ferner ermöglicht das System **500** die Priorisierung unter bandbreitenbedarfsreduzierten TCP-Verbindungen für den Zugriff auf TCP-Spoofing-Ressourcen (z.B. die verfügbare Bandbreite und den verfügbaren Pufferspeicherplatz). diese Priorisierung wird für alle Arten von Verkehr verwendet, die um Systemressourcen konkurrieren.

[0109] Im Hinblick auf die Backbone-Verbindung eignet sich das System **500** für die Anwendung in einem Satellitennetzwerk, wie dem WAN. Das heißt, das Backbone-Protokoll ist für die Satellitenanwendung optimiert, da Steuerblock-Ressourcenanforderungen minimiert sind, und eine wirksame Fehlerentdeckung für fallen gelassene Pakete bereitgestellt wird. Das System **500** stellt auch einen Feedback-Mechanismus bereit, um die maximale Pufferspeicherplatz-Ressourceneffizienz zu unterstützen. Ferner stellt das System **500** einen verringerten Bestätigungsverkehr durch Verwenden eines einzigen Backbone-Protokolls ACK zur Bestätigung der Daten von mehreren TCP-Verbindungen bereit.

[0110] Fig. 6 stellt den Fluss von IP-Paketen durch einen PEP-Endpunkt gemäß einer Ausführungsform der vorliegenden Erfindung dar. Wenn IP-Pakete an der lokalen LAN-Schnittstelle **220** empfangen werden, bestimmt der PEP-Endpunkt **210** (wie vom Entscheidungspunkt A dargestellt), ob die Pakete für einen Host bestimmt sind, der sich vor Ort befindet; falls dies der Fall ist, werden die IP-Pakete zur richtigen lokalen LAN-Schnittstelle **220** weitergeschickt. Falls die IP-Pakete für einen entfernten Host bestimmt sind, dann entscheidet der PEP-Endpunkt **210** am Entscheidungspunkt B, ob der Verkehr ein TCP-Segment ist. Falls der PEP-Endpunkt **210** bestimmt, dass die Pakete tatsächlich TCP-Segmente sind, dann bestimmt der TSK **280**, ob der Bandbreitenbedarf der TCP-Verbindung reduziert werden soll. Falls der PEP-Endpunkt **210** bestimmt, dass die Pakete keine TCP-Segmente sind, dann verarbeitet der BPK **282** den Verkehr zusammen mit dem PK **284** und dem PSK **286** für die letztendliche Übertragung an das WAN. Es sei darauf hingewiesen, dass der BPK **282** keine nicht-bandbreitenbedarfsreduzierten IP-Pakete verarbeitet; d.h. die Pakete fließen direkt zum PD 284. Wie in Fig. 6 dargestellt, wird Verkehr, der von der WAN-Schnittstelle **230** empfangen wird, untersucht, um zu bestimmen, ob der Verkehr ein richtiges PBP-Segment (Entscheidungspunkt D) für

den speziellen PEP-Endpunkt **210** ist; falls die Entscheidung positiv ausfällt, dann werden die Pakete zum BPK **282** und dann zum TSK **280** geschickt.

[0111] Die Routing-Unterstützung schließt ein Routing zwischen den Ports des PEP-Endpunkts **210** (Fig. 2) ein, z.B. von einem Multimedia VASAT LAN-Port zu einem anderen. Architektonisch passen die Funktionen des TCP-Spoofing, der Priorisierung und der Pfadauswahl zwischen die IP-Routing-Funktion und das WAN. Die PEP-Funktion muss nicht auf IP-Pakete angewendet werden, die von lokalem Port zu lokalem Port innerhalb des gleichen PEP-Endpunkts **210** geführt werden. TCP-Spoofing, Priorisierung und Pfadauswahl werden auf IP-Pakete angewendet, die von einer lokalen PEP-Endpunktschnittstelle empfangen werden und die von der Routing-Funktion als für eine andere Site bestimmt bestimmt wurden.

[0112] Fig. 7 zeigt die Beziehung zwischen PEP-Endpunkten und PEP-Endpunktprofilen gemäß einer Ausführungsform der vorliegenden Erfindung. PEP-Parameter werden in erster Linie über einen Satz von Profilen **701** und **703** konfiguriert, die mit einem oder mehreren PEP-Endpunkten **705** assoziiert sind. In einem Ausführungsbeispiel werden PEP-Parameter auf einer PEP-Endpunktbasis konfiguriert, so als ob TCP-Spoofing global möglich wäre Diese Parameter werden in den PEP-Endpunktprofilen **701** und **703** konfiguriert. Es sei darauf hingewiesen, dass Parameter, die sich auf bestimmte PEP-Kernels beziehen, über andere Arten von Profilen konfiguriert werden können. Die Profile **701** und **703** sind ein Netzwerkverwaltungskonstrukt; intern verarbeitet ein PEP-Endpunkt **705** einen Satz Parameter, die über eine oder mehrere Dateien empfangen werden.

[0113] Sobald der PEP-Endpunkt **705** neue Parameter empfängt, vergleicht die Plattformumgebung die neuen Parameter mit den vorhanden Parametern, findet heraus, welche der PEP-Kernels von den Parameteränderungen beeinflusst werden und gibt dann die neuen Parameter an die beeinflussten Kernels weiter. In einer Beispielsumgebung werden alle Parameter dynamisch installiert. Mit Ausnahme von Parametern, die komponentenspezifisch sind (wie die IP-Adressen einer Komponente), können alle Parameter mit Default-Werten definiert werden.

[0114] Wie bereits erwähnt, kann der PEP-Endpunkt **210** in einer Reihe von verschiedenen Plattformen gemäß den verschiedenen Ausführungsformen der vorliegenden Erfindung implementiert werden. Diese Plattformen schließen ein IP-Gateway, ein Multimedia-Relais, einen Multimedia VSAT (Very Small Aperture Terminal) und eine Personal Earth Station (PES) Remote ein, wie jeweils in den Fig. 8–Fig. 11 dargestellt. Im Allgemeinen definiert der PEP-Endpunkt **210**, wie in Fig. 2 dargestellt, eine lokale LAN-Schnittstelle **220**, durch welche Schnittstelle der PEP-Endpunkt **210** die IP-Hosts, die an der Site angesiedelt sind, miteinander verbindet. Eine WAN-Schnittstelle **230** ist eine Schnittstelle, durch die der PEP-Endpunkt **210** Verbindungen zu anderen Sites herstellt. Es sei darauf hingewiesen, dass eine WAN-Schnittstelle **230** physisch ein LAN-Port sein kann. Die Fig. 8–Fig. 11 beschreiben nachstehend spezifische LAN- und WAN-Schnittstellen der verschiedenen PEP-Endpunktplattformen. Welche LAN- und WAN-Schnittstellen jeweils verwendet werden, hängt davon ab, welche Fern-Site-PEP-Endpunkte verwendet werden, welche Konfiguration der Hub und die Fern-Site-PEP-Endpunkte haben und welche Pfadauswahlregeln konfiguriert den können.

[0115] Fig. 8 zeigt die Schnittstellen des PEP-Endpunkts, der als ein IP-Gateway implementiert ist, gemäß einer Ausführungsform der vorliegenden Erfindung. Beispielsweise weist ein IP-Gateway **801** eine einzige lokale LAN-Schnittstelle auf, bei der es sich um eine Unternehmensschnittstelle **803** handelt. Das IP-Gateway **803** verwendet zwei WAN-Schnittstellen **805**, um IP-Pakete von Fern-Site-PEP-Endpunkten zu empfangen und an diese zu senden: eine Backbone-LAN-Schnittstelle und eine Wide Area Access (WAA)-Lan-Schnittstelle.

[0116] Die Backbone-LAN-Schnittstelle **805** wird verwendet, um beispielsweise über ein Satelliten-Gateway (SGW) und eine VSAT-Outroute IP-Pakete an Fern-Site-PEP-Endpunkte zu senden. Eine VSAT-Outroute kann direkt von Multimedia-Relais (Fig. 9) und Miltimedia-VSATs (Fig. 10) empfangen werden (und ist der vorherrschende Pfad, der mit diesen Endpunkten verwendet wird); jedoch können IP-Pakete auch über eine VSAT-Outroute an ein PES Remote (Fig. 11) geschickt werden.

[0117] Fig. 9 zeigt eine Multimedia-Relais-Implementierung eines PEP-Endpunkts gemäß einer Ausführungsform der vorliegenden Erfindung. Ein Multimedia-Relais weist zwei oder drei lokale LAN-Schnittstellen **903** auf. Zusätzlich weist das Multimedia-Relais **901** bis zu zwei WAN-Schnittstellen **905** auf, um IP-Pakete an Hub-Site-PEP-Endpunkte zu schicken: eine seiner LAN-Schnittstellen und eine PPP-Serienportschnittstelle, und vier oder fünf Schnittstellen zum Empfangen von IP-Paketen von Hub-Site-PEP-Endpunkten, einer VSAT-Outroute, allen seinen LAN-Schnittstellen und einer PPP-Serienport-Schnittstelle. Es sei darauf hingewiesen, dass eine PPP (Punkt-zu-Punkt-Protokoll)-Serienportschnittstelle und eine LAN-Schnittstelle im Allgemeinen nicht gleichzeitig verwendet werden.

[0118] Ein Multimedia-Relais **901** unterstützt die Verwendung aller seiner LAN-Schnittstellen **903**, um gleichzeitig IP-Pakete von Hub-Site-Endpunkten zu senden und von diesen zu empfangen. Ferner unter-

stützt ein Multimedia-Relais **905** die Verwendung einer VADB (VPN Automatic Dial Backup)-Serienportschnittstelle zum Senden und Empfangen von IP-Paketen an die und von den Hub-Site-PEP-Endpunkten.

[0119] Fig. 10 zeigt eine Multimedia-VSAT-Implementierung des PEP-Endpunkts gemäß einer Ausführungsform der vorliegenden Erfindung. Ein Multimedia-VSAT **1001** weist in einem Ausführungsbeispiel zwei lokale LAN-Schnittstellen **1003** auf. Es kann Unterstützung für eine oder mehrere lokale PPP-Serienportschnittstellen verwendet werden. Der Multimedia-VSAT **1001** weist zwei WAN-Schnittstellen **1005** zum Versenden von IP-Paketen zu Hub-Site PEP-Endpunkten auf: eine VSAT-Inroute und eine ihrer LAN-Schnittstellen. Der Multimedia-VSAT **1001** weist somit drei Schnittstellen zum Empfangen von Daten von Hub-Site-PEP-Endpunkten auf, die VSAT-Outroute und ihre beiden LAN-Schnittstellen **1003**. Ein Multimedia VSAT **1003** kann die gleichzeitige Verwendung seiner beiden LAN-Schnittstellen **1003** zum Senden und Empfangen von IP-Paketen an die und von den Hub-Site-PEP-Endpunkte(n) unterstützen. Der Multimedia VSAT **1003** unterstützt ferner die Verwendung einer VADB-Serienportschnittstelle zum Senden und Empfangen von IP-Paketen an die und von den Hub-Site-PEP-Endpunkte(n).

[0120] Fig. 11 zeigt eine PES-Remote-Implementierung eines PEP-Endpunkts gemäß einer Ausführungsform der vorliegenden Erfindung. Ein PES-Remote **1101** kann eine lokale LAN-Schnittstelle und/oder mehrere lokale IP (z.B. PPP, SLIP usw.) Serienportschnittstellen aufweisen, die zusammen als LAN-Schnittstellen **1103** bezeichnet werden. Die jeweiligen LAN-Schnittstellen **1103** hängen von der speziellen PES-Remote-Plattform ab. Der PES-Remote **1101** weist in einem Ausführungsbeispiel bis zu fünf WAN-Schnittstellen **1105** zum Senden von IP-Paketen an Hub-Site-Endpunkte, eine ISBN-Inroute, eine LAN-Schnittstelle, eine VADB-Serienportschnittstelle, eine Frame-Relais-Serienportschnittstelle und eine IP-Serienport-Schnittstelle, und bis zu fünf vorhandene Schnittstellen zum Empfangen von IP-Paketen von Hub-Site-PEP-Endpunkten auf: eine ISBN-Outroute, eine LAN-Schnittstelle, eine VADB-Serienportschnittstelle, eine Frame-Relais-Serienportschnittstelle und eine IP-Serienportschnittstelle auf. Die physische Frame Relais-Serienportschnittstelle kann mehrere permanente virtuelle Verbindungen bzw. Permanent Virtual Circuits (PVCs) unterstützen, von denen einige zu lokalen Schnittstellen **1103** äquivalent sind und von denen einige WAN-Schnittstellen sind.

[0121] Fig. 12 zeigt den Fluss von TCP-Spoofing-Puffern durch einen PEP-Endpunkt gemäß einer Ausführungsform der vorliegenden Erfindung. In diesem Bespiel sind sechs logische Puffer-Pools mit dem Empfangen, Verarbeiten und Weitersenden von TCP-Segmenten für bandbreitenbedarfreduzierte TCP-Verbindungen befasst: ein LAN-zu-WAN (L2W)-Puffer-Pool **1201**; ein WAN-zu-LAN (W2L)-Puffer-Pool **1203**; ein LAN-Empfangs- (LAN Rx-) Puffer-Pool **1205**; ein LAN-Übermittlungs- (LAN Tx-) Puffer-Pool **1207**; ein WAN-Empfangs- (WAN Rx-) Puffer-Pool **1209** und ein WAN-Übermittlungs- (WAN Tx-) Puffer-Pool **1211**.

[0122] Die in Fig. 12 dargestellten Schnittstellen und Puffer-Pools sind logische Instanzen. Es sei darauf hingewiesen, dass der in Fig. 12 dargestellte Pufferfluss in manchen Fällen zum Zwecke der Erläuterung vereinfacht ist; beispielsweise kann „ein Puffer" aus mehreren physischen Puffern bestehen. Physisch können mehr als eine LAN- oder WAN-Schnittstelle vorhanden sein, und in manchen Fällen kann für manche Plattformen die gleiche physische Schnittstelle sowohl als LAN-Schnittstelle **1213** als auch als WAN-Schnittstelle **1215** verwendet werden. Die Puffer-Pools **1201**, **1203**, **1205**, **1207**, **1209** und **1211** sind insofern logisch, als der gleiche physische Puffersatz verwendet werden kann, um mehr als einen der Puffer-Pools zu implementieren, entweder um die Implementierung zu erleichtern oder weil die LAN- und WAN-Schnittstellen **1213**, **1215** die gleiche physische Schnittstelle sind. Einzelheiten über die plattformspezifische physische Implementierung von logischen Puffer-Pools **1201**, **1203**, **1205**, **1207**, **1209** und **1211** sind nachstehend beschrieben.

[0123] Wenn ein IP-Paket von dem lokalen LAN ankommt, empfängt die LAN-Schnittstelle **1213** das Paket in einem Pufferspeicher vom LAN Rx-Puffer-Pool **1205** und gibt das Paket an die Plattformumgebung **210** weiter. Die Plattformumgebung **210** kopiert das IP-Paket aus dem LAN Rx-Puffer **1205** in einen LAN-zu-WAN-Puffer **1201** und schickt dann den LAN Rx-Puffer **1205** zur LAN-Schnittstelle **1213** zurück. In einer Plattform, wo der LAN-Rx-Puffer **1205** und der LAN-zu-WAN-Puffer **1201** physisch identisch sind, kann die Umgebung **210** das Kopieren vermeiden und einfach einen LAN-zu-WAN-Puffer **1201** gegen den LAN Rx-Puffer **1205** tauschen. Ob nun tatsächlich eine Kopie erstellt wird oder nicht, falls kein LAN-zu-WAN-Puffer verfügbar ist, wird das IP-Paket verworfen (durch Zurückschicken des ursprünglichen LAN Rx-Puffers **1205** an die LAN-Schnittstelle **1213**) und muss von dieser auf die gleiche Weise aufgenommen werden als würde das IP-Paket das LAN durchlaufen.

[0124] Die Umgebung **210** gibt IP-Pakete, die bandbreitenbedarfreduzierte TCP-Segmente enthalten, an den TCP-Spoofing-Kernel **280** weiter (wenn ein TCP-Spoofing möglich ist). Der LAN-zu-WAN-Puffer **1201**, der IP-Pakete handhabt, die keine TCP-Segmente enthalten, ist nachstehend beschrieben. Die

Umgebung **210** erkennt ein TCP-bandbreitenbedarfsreduziertes TCP-Segmente durch die Anwesenheit eines CCB für das Segment. Die Umgebung **210** gibt auch TCP <SYN>-Segmente an den TSK **280** weiter, um zu bestimmen, ob der Bandbreitenbedarf einer neuen Verbindung reduziert werden soll. Falls das TCP <SYN>-Segment nicht zu einer TCP-Verbindung gehört, deren Bandbreitenbedarf reduziert werden soll, schickt der TSK **280** das IP-Paket mit einem Hinweis, das TCP-Segment ohne Bandbreitenbedarfsreduzierung weiterzuschicken, zurück zur Umgebung **210**. Es gibt auch Umstände, unter denen der TSK **280** ein TCP-Segment zurückschickt, um es ohne Bandbreitenbedarfsreduzierung weiterschicken zu lassen, wenn ein CCB für die TCP-Verbindung vorhanden ist. Falls das TCP-Segment zu einer TCP-Verbindung gehört, deren Bandbreitenbedarf gerade reduziert wird (oder gleich reduziert werden soll), verarbeitet der TSK **280** das TCP-Segment und schickt dann entweder den Inhalt des TCP-Segments zu seinem TSK 280-Peer oder verwirft ihn und schickt den Puffer des Segments zur Plattformumgebung **210** zurück. Die Plattformumgebung **210** schickt ihrerseits den Puffer zum LAN-zu-WAN-Puffer-Pool zurück. In einigen Fällen muss der TSK **280** das empfangene TCP-Segment nicht weiterschicken, sondern muss nur eine TSK-Nachricht (als Folge des Empfangs des TCP-Segments) an seinen TSK-Peer schicken. Wenn beispielsweise ein TCP <SYN>-Segment empfangen wird, wird das <SYN>-Segment nicht an den TSK-Peer weitergeschickt, sondern es kann sein, dass eine Connection Request-Nachricht an den TSK-Peer geschickt werden muss). Wenn dies der Fall ist, benutzt der TSK **280** einfach den Puffer erneut, indem das TCP-Segment empfangen wurde, statt den Puffer des TCP-Segments zu verwerfen und dann um einen neuen Puffer zu bitten, um die zu versendende TSK-Nachricht zu erzeugen.

[0125] In Fällen, wo der TSK **280** eine TSK-Nachricht an seinen Peer asynchron zum Empfang eines TCP-Segments verschicken muss, fordert der TSK **280** einen LAN-zu-WAN-Puffer **1201** von der Plattformumgebung **210** an und nutzt diesen Puffer **1201**, um die Nachricht zu konstruieren. Um eine Daten- oder Steuer-TSK-Nachricht an seinen TSK-Peer zu schicken, gibt der TCP-Spoofing-Kernel **280** den Puffer der Nachricht (zusammen mit einem Hinweisen, welche Backbone-Verbindung zum Senden der Nachricht benutzt werden soll) an den Backbone Protocol Kernel **282** weiter. Sobald eine Nachricht an den BPK **282** geschickt wurde, übernimmt der BPK **282** das Eigentum bzw. die Inhaberschaft über den LAN-zu-WAN-Puffer **1201** der Nachricht. TSK-Nachrichten werden vom BPK **282** zu seinem BPK-Peer als PBP-Segmente geschickt. Um ein PBP-Segment zu verschicken, gibt der BPK **282** das Segment als IP-Paket an die Plattformumgebung **210** weiter, um es an die geeignete WAN-Schnittstelle **1215** zu übermitteln. Die Umgebung **210** gibt das IP-Paket an die geeignete WAN-Schnittstelle **1215** weiter, wobei der LAN-zu-WAN-Puffer **1201** in einen WAN Tx-Puffer **1211** kopiert wird.

[0126] Weil der BPK **282** für die garantierte Zustellung von TSK-Nachrichten sorgen muss, muss der BPK **282** jede TSK-Nachricht, die er übermittelt, zurückholen und für eine potentielle erneute Sendung aufbewahren. Daher muss die Plattformumgebung **210** (wenn über eine Flagge bzw. ein Flag, das mit der Schnittstelle verwendet wird, aufgefordert) ein an sie geschicktes IP-Paket nach dessen Übermittlung zurück zum BPK **282** schicken. Es sei darauf hingewiesen, dass, wenn die Umgebung **210** IP-Pakete für eine bestimmte Backbone-Verbindung zum BPK **282** zurückschickt, die Umgebung die IP-Pakete in der Reihenfolge, in der sie sie vom BPK **282** erhalten hat, an diesen zurückschicken muss. Gemäß einem Ausführungsbeispiel kann dies automatisch durch Ausführen einer sofortigen Kopie in einen WAN Tx-Puffer geschehen. Alternativ dazu kann dies durch Verwendung eines Queuing-Mechanismus geschehen, um sicherzustellen, dass die Pakete in der richtigen Reihenfolge zurückgeschickt werden. In einer Plattform **210**, die einen LAN-zu-WAN-Puffer **1201** und einen WAN Tx-Puffer **1211** nutzt, die kompatibel sind, muss die Umgebung **210** auch gar keine Kopie erstellen, falls der BPK **282** das IP-Paket nicht zurückfordert. Falls die Puffer **201** und **1211** kompatibel sind, kann der zugeordnete WAN Tx-Puffer **1211** zum LAN-zu-WAN-Puffer-Pool **1201** zurückgeschickt werden, wobei der LAN-zu-WAN-Puffer **1201** als WAN Tx-Puffer **1211** weitergeschickt wird.

[0127] Der Backbone Protocol Kernel **282** kann auch Segmente erzeugen, die an seinen BPK-Peer geschickt werden sollen, ohne eine Nachricht vom TSK **280** zu empfangen, z.B. um eine Bestätigung für empfangene PBP-Segmente zu verschicken. Um ein solches Segment zu verschicken, ordnet der BPK **282** einen Puffer aus dem LAN-zu-WAN-Puffer-Pool **1201** (über die Plattformumgebung **210**) zu, konstruiert das PBP-Segment, das er senden muss und schickt dann das Segment als IP-Paket auf die gleiche Weise wie er PBP-Segmente, die TSK-Nachrichten enthalten, versendet, an die Plattformumgebung **210**. Es sei darauf hingewiesen, dass die Zuordnung von Puffern, um PBP-Bestätigungen zu verschicken, unabhängig von dem Empfang von PBP-Segmenten stattfindet. Der BPK **282** verarbeitet auch dann jedes empfangene PBP-Segment, wenn kein LAN-zu-WAN-Puffer **1201** verfügbar ist, um eine Antwort an das Segment zu schicken. Das Fehlen eines Puffers, um eine Antwort zu verschicken, wird einfach auf die gleiche Weise gehandhabt, als wenn das Segment erfolgreich übermittelt worden wäre, aber beim Durchgang durch das WAN verloren gegangen wäre. Nachdem der Backbone-Kernel mit einem von ihm übertragenen Segment fertig ist, z.B. eine Bestä-

tigung für das Segment von seinem BPK-Peer erhalten hat, gibt er den Puffer des Segments an den LAN-zu-WAN-Puffer-Pool zurück.

[0128] Der Verlust eines empfangenen oder übertragenen TCP-Segments oder PBP-Segments, weil kein Puffer verfügbar ist, ist nicht kritisch. Das verlorene IP-Paket kann auf die gleiche Weise wiedererlangt werden als wenn das IP-Paket während des Durchgangs durch das LAN oder WAN verloren gegangen wäre. Das Unvermögen, eine TSK-Nachricht zu senden, weil kein Puffer zur Verfügung steht, ist jedoch eine ernstere Situation. Der TSK **280** nimmt an, dass Nachrichten in der Leitung, die zwischen ihm und seinem Peer durch das PEP-Backbone-Protokoll bereitgestellt werden, nicht verloren gehen können. Daher ist ein spezielles Handeln erforderlich, wenn der TSK **280** versucht, eine TSK-Nachricht von Grund auf zu erzeugen und dies nicht tun kann. In manchen Fällen, beispielsweise beim Erzeugen einer TSK-Peer-Parameternachricht, besteht die angemessene Reaktion im Starten eines Zeitnehmers und dem erneuten Versuch, die Nachricht zu versenden, wenn der Zeitnehrmer abgelaufen ist. In anderen Fällen, beispielsweise bei Unfähigkeit, eine Connection Terminated-Nachricht zu versenden, könnte die angemessene Reaktion die Ignorierung des Ereignisses sein, das ein Versenden der CT-Nachricht erfordert hat. Wenn die Nachricht beispielsweise aufgrund eines Time-Out verschickt wird, kann der Zeitgeber mit irgendeinem kleinen Wert erneut gestartet werden und neu verarbeitet werden, wenn dieser wieder abgelaufen ist.

[0129] Wenn ein IP-Paket vom WAN ankommt, empfängt die WAN-Schnittstelle **1215** das Paket in einem Puffer vom WAN Rx-Puffer-Pool **1209** und gibt es an die Plattformumgebung **210** weiter. Die Plattformumgebung **210** kopiert das IP-Paket aus dem WAN-Rx-Puffer **1209** in einen WAN-zu-LAN-Puffer **1203** und schickt dann den WAN Rx-Puffer **1209** an die WAN-Schnittstelle **1215** zurück. In einer Plattform **210**, in der der WAN Rx-Puffer **1209** und der WAN-zu-LAN-Puffer **1203** physisch identisch sind, kann die Umgebung **210** das Kopieren vermeiden und einfach einen WAN-zu-LAN-Puffer **1203** gegen den WAN Rx-Puffer **1209** tauschen. Ob es nun zu einer tatsächlichen Kopie kommt oder nicht, das IP-Paket wird verworfen (durch Zurückschicken des originalen WAN Rx-Puffers **1209** an die WAN-Schnittstelle), wenn kein WAN-zu-Lan-Puffer verfügbar ist, und muss auf die gleiche Weise wiederhergestellt werden als wenn das IP-Paket beim Durchqueren des WAN verloren gegangen wäre. Die Umgebung **210** gibt alle IP-Pakete, die PBP-Segmente (adressiert an diesen PEP-Endpunkt **210**) enthalten, an den Backbone Protocol Kernel **282** zurück. Der WAN-zu-LAN-Puffer, der andere Arten von IP-Paketen handhabt, ist nachstehend beschrieben.

[0130] Wie der BPK mit PBP-Segmenten umgeht, hängt von der Art des PBP-Segments ab. Was den Umgang mit Puffern betrifft, so gibt es zwei Arten von PBP-Segmenten: (1) PBP-Segmente, die sofort verarbeitet und verworfen werden können, d.h. PBP-Steuersegmente; und (2) PBP-Segmente, die zum TCP-Spoofing-Kernel **280** weitergeschickt werden müssen, d.h. TSK-Nachrichten. Für ein PBP-Steuersegment, z.B. ein PBP-Segment, das verwendet wird, um Backbone-Verbindungen einzurichten, kann der Backbone Protocol Kernel **282** jede vom Segment benötigte Aktion durchführen und den Puffer des Segments dann zum WAN-zu-LAN-Puffer-Pool **1203** zurückschicken. Der BPK **282** schickt empfangene TSK-Nachrichten an den TCP-Spoofing Kernel **280**. Sobald der BPK **282** eine Nachricht an den TSK **280** geschickt hat, übernimmt der TSK **280** die Inhaberschaft über den WAN-zu-LAN-Puffer **1203** der Nachricht. Der Umgang des TSK mit dem WAN-zu-LAN-Puffer ist nachstehend beschrieben. Es sei darauf hingewiesen, dass ein Segment, das eine TSK-Nachricht enthält, nicht unbedingt sofort an den TSK **280** weitergeschickt werden muss. Segmente mit der falschen Reihenfolge werden vom BPK **282** in einer Resequenzierungs-Warteschlange gehalten, während der BPK **282** auf die fehlenden Elemente wartet. (Der BPK muss TSK-Nachrichten in der richtigen Reihenfolge an den TCP-Spoofing-Kernel schicken). Außerdem erzeugt der Backbone Protocol Kernel keine Nachrichten, um Informationen (z.B. Backbone-Verbindungs-Resets) an den TCP-Spoofing-Kernel zu schicken. Jede Information, die der BPK **282** zum TSK **280** weiterschicken muss, wird anhand einer prozeduralen Schnittstelle verschickt. Daher muss der BPK **282** niemals einen WAN-zu-LAN-Puffer **1203** für sich selbst reservieren.

[0131] Der TCP-Spoofing-Kernel **280** empfängt zwei Arten von Nachrichten von seinem TSK-Peer: Steuernachrichten (z.B. Connection Request-Nachrichten) und Datennachrichten (z.B. TCP-Data-Nachrichten). Beide Nachrichtenarten können in manchen Fällen sofort von der TSK **280** fallen gelassen werden (beispielsweise nach Empfang einer TCP-Nachricht für eine Verbindung, die nicht mehr besteht). Dies wird dadurch bewerkstelligt, dass der Puffer der Nachricht einfach zum WAN-zu-LAN-Puffer-Pool **1203** zurückgeschickt wird. Im Allgemeinen ist jedoch eine Verarbeitung für eine Nachricht, die von einem TSK-Peer erhalten wird, notwendig. Steuernachrichten können die Erzeugung eines entsprechenden TCP-Segments, das an einen lokalen Host geschickt wird, erfordern. Beispielsweise hat der Empfang einer Connection Request-Nachricht im Allgemeinen zur Folge, dass ein TCP <SYN>-Segment an einen lokalen Host geschickt wird. Der Empfang einer Connection Established-Nachricht führt jedoch nicht dazu, dass ein TCP <SYN,ACK>-Segment zu einem lokalen Host geschickt wird, wenn der TSK **280** das <SYN,ACK>-Segment bereits verschickt

hat. Wenn eine Steuernachricht verlangt, dass ein TCP-Segment an einen lokalen Host geschickt wird, speichert der TSK **280** sämtliche Informationen, die er benötigt, aus der Steuernachricht und nutzt dann den WAN-zu-LAN-Puffer **1203** der Steuernachricht, um das TCP-Segment zu konstruieren, das versendet werden muss. Abgesehen davon, dass sie effizienter ist, vermeidet die Wiederverwendung des WAN-zu-LAN-Puffers **1203** Fehlerszenarios, wo kein zusätzlicher WAN-zu-LAN-Puffer **1203** für das TCP-Segment, das erzeugt werden muss, zur Verfügung steht. Für eine Datennachricht muss der TCP-Spoofing-Kernel zuerst die TSK-Nachricht in ein TCP-Datensegment umwandeln. Dies wird in erster Linie durch Ersetzen der PBP- und TSK-Puffer-Header mit einem geeigneten TCP-Header **1515** unter Verwendung des nachstehend beschriebenen Mechanismus bewerkstelligt.

[0132] Nachdem der TCP-Spoofing-Kernel **280** eine TSK-Nachricht in ein TCP-Segment umgewandelt hat, schickt der TSK **280** das TCP-Segment an einen lokalen Host, indem er das Segment als IP-Paket an die Plattformumgebung **210** zur Übertragung an die geeignete LAN-Schnittstelle **1213** schickt. Die Umgebung **210** schickt das IP-Paket zur Übertragung an die LAN-Schnittstelle **1213**; dies wird durch Zuordnen eines LAN Tx-Puffers **1207** und anschließendes Kopieren des IP-Pakets aus dem WAN-zu-LAN-Puffer **1203** in den LAN-Tx-Puffer **1207** bewerkstelligt. Eine Kopie wird erstellt, weil der TSK **280** für eine garantierte Zustellung der TCP-Datensegmente sorgen muss und daher viele der TCP-Datensegmente, die der TSK **280** überträgt, für eine mögliche Neuübertragung zurückholen und halten muss. Daher schickt die Umgebung **210** (wenn über ein mit der Schnittstelle verwendetes Flag aufgefordert) die IP-Pakete, die ihr zugeschickt wurden, zu TSK **280** zurück, nachdem diese Pakete übermittelt wurden. Das Kopieren des IP-Pakets in einen LAN Tx-Puffer **1207** erlaubt es der Umgebung **210**, dies sofort durchzuführen. Falls die Umgebung **210** keinen LAN Tx-Puffer **1207** zuordnen kann, um das IP-Paket in diesen zu kopieren, muss die Umgebung **210** das IP-Paket zum TSK **280** zurückschicken, als ob das IP-Paket übermittelt worden wäre. Der TSK **280** gleicht diesen Fehler dann auf die gleiche Weise aus als wenn das Paket beim Durchgang durch das lokale LAN verloren gegangen wäre. Es sei darauf hingewiesen, dass, wenn die Umgebung **210** IP-Pakete für eine bestimmte Verbindung an den TSK **280** zurückschickt, die Umgebung **210** die IP-Pakete in der Reihenfolge, in der sie ihr vom TSK **280** zugeschickt wurden, an den TSK **280** zurückschicken muss. Die sofortige Kopie macht die Erfüllung dieser Forderung einfach.

[0133] Der TCP-Spoofing-Kernel **280** kann auch TCP-Segmente erzeugen, die zu einem lokalen Host geschickt werden sollen, ohne eine Nachricht von seinem TSK-Peer zu empfangen, z.B. um eine Bestätigung für empfangene TCP-Datensegmente zu verschicken. Um ein solches Segment zu versenden, reserviert der TSK **280** einen Puffer von einem WAN-zu-LAN-Puffer-Pool **1203**, konstruiert das TCP-Segment, das der TSK **280** versenden muss und schickt dann das Segment auf die gleiche Weise, wie er TCP-Segmente, die von einer TSK-Nachricht erzeugt werden, die er von seinem TSK-Peer empfangen hat, als IP-Paket an die Plattformumgebung **210**. Es sei darauf hingewiesen, dass die Reservierung von Puffern, um TCP-Datenbestätigungen zu versenden, unabhängig vom Empfang von TCP-Segmenten abläuft. Der TSK verarbeitet auch dann alle empfangenen TCP-Segmente, einschließlich von Datensegmenten, wenn kein WAN-zu-LAN-Puffer verfügbar ist, um eine Antwort auf das Segment zu schicken. Das Fehlen eines Puffers, um eine Antwort zu senden, wird einfach auf dieselbe Weise überwunden als ob das Segment erfolgreich übertragen worden wäre, aber beim Durchgang durch das LAN verloren gegangen wäre. Nachdem der TCP-Spoofing-Kernel mit einem Segment, das er übertragen hat, fertig ist, z.B. eine Bestätigung für das Segment vom lokalen Host empfangen hat, schickt er den Puffer des Segments zum WAN-zu-LAN-Puffer-Pool zurück.

[0134] Fig. 13 zeigt ein Diagramm des Puffer-Managements für nicht-bandbreitenbedarfsreduzierte TCP-Verbindungen und für Nicht-TCP (z.B. UDP)-Verkehr gemäß der vorliegenden Erfindung. Das Puffer-Management im Fall ohne Reduzierung des Bandbreitenbedarfs ähnelt dem Puffer-Management für bandbreitenbedarfsreduzierte TCP-Verbindung, ist aber viel einfacher. Wie in Fig. 13 dargestellt, kopiert in der LAN-nach-WAN-Richtung die Plattformumgebung **210** empfangene IP-Pakete aus LAN Rx-Puffern **1205** in LAN-zu-WAN-Puffer **1201**. Nicht-TCP-IP-Pakete werden direkt an die WAN-Schnittstelle **1215** weitergeschickt, ohne durch den TCP-Spoofing-Kernel oder den Backbone Protocol Kernel fließen zu müssen. Nicht-bandbreitenbedarfsreduzierte TCP-IP-Pakete werden wie Nicht-TCP-IP-Pakete weitergeschickt, nachdem der TSK **280** sie „zurückweist". (Falls das Spoofing global nicht möglich ist, macht sich die Umgebung **210** nicht die Mühe, die TCP-IP-Pakete durch den TSK **280** zu schicken.) In der Richtung von WAN nach LAN ist das Verfahren ähnlich. Die Plattformumgebung **210** kopiert empfangene IP-Pakete aus WAN Rx-Puffern **1209** in WAN-zu-LAN-Puffer **1203** und schickt dann, bei allen IP-Paketen, die keine PBP-IP-Pakete sind, die eine der IP-Adressen der Plattform als Zieladresse haben, die Pakete an die (geeignete) LAN-Schnittstelle **1213**, wobei sie die IP-Pakete in LAN Tx-Puffer **1207** kopiert. In einigen Plattformen kann es möglich sein, dass die Plattformumgebung **210** die IP-Pakete direkt aus WAN Rx- in LAN Tx-Puffer kopiert. Diese Pakete müssen nicht von einem PEP-Kernel verarbeitet werden.

[0135] Die Backbone-Verbindung, die mit einem Puffer assoziiert ist, wird im Puffer gespeichert. Wenn keine Backbone-Verbindung mit dem Puffer assoziiert ist, wird ein Wert 0 × FFFF verwendet. Zum Zwecke der Fehlerbeseitigung (und um den Pufferbehandlungs-Code symmetrisch zu halten) kann die Plattformumgebung **210** die Zahl der aktuell zugeordneten Puffer, die mit „Backbone-Verbindung" 0 × FFFF assoziiert sind, verfolgen.

[0136] Fig. 14 ist ein Diagramm eines Grundformats des Puffers, der verwendet wird, um die PEP-Funktion zu implementieren, gemäß einer Ausführungsform der vorliegenden Erfindung. Ein Puffer **1400** schließt einen Puffer-Header **1401** ein, der plattformspezifische Pufferfelder enthält, falls solche Felder vorhanden sind. Das Format (und selbst die Existenz) dieser Felder ist nur der Plattformumgebung **210** bekannt. An den plattformspezifischen Puffer-Header **1401** schließt sich ein allgemeiner PEP-Puffer-Header **1403** an, der in einem Ausführungsbeispiel eine Länge von etwa 30 bis 44 Byte aufweist. Die Felder in diesem Header **1403** sind den PEP-Kernels bekannt und werden von diesen genutzt. Der Puffer **1400** schließt auch einen Abschnitt ein, der für das IP-Paket **1405** vorgesehen ist, und der zusätzlich zum allgemeinen PEP-Puffer-Header **1403** den „Nutzinhalt" des Puffers **1400** darstellt.

[0137] Die Puffer **1400** werden über einen Zeiger auf den Anfang des allgemeinen PEP-Puffer-Headers **1401** von der Umgebung **210** zu einem Kernel weitergegeben, von einem Kernel zu einem anderen Kernel und von einem Kernel zur Umgebung **210**. Etwaige Zeigeranpassungen, die nötig sind, um einem plattformspezifischen Puffer-Header **1401** gerecht zu werden, werden von der Plattformumgebung **210** vorgenommen.

[0138] Die Plattformumgebung **210** stellt gemäß einem Ausführungsbeispiel den Aufgabenkontext bereit, in dem die PEP-Kernels arbeiten. Unter dem Gesichtspunkt des plattformspezifischen Puffer-Managements ist daher die Plattformumgebung **210** der ausdrückliche Inhaber aller Puffer, die der Verwendung für die PEP-Funktion zugeordnet sind. Puffer-Behandlungsformalitäten in Bezug auf (explizite) Inhaberschaft über die Puffer (falls eine für eine bestimmte Plattform existiert) tauchen an dem Punkt auf, wo die Plattformumgebung **210** einen Puffer von außerhalb des PEP-Kontext empfängt oder dorthin schickt. Innerhalb des Kontexts der Aufgabe der Plattformumgebung **210** wird angenommen, dass ein Puffer von dem Kernel innegehabt wird, der ihn gerade besitzt. Es muss jedoch kein formaler Pufferinhaberschaftstransfer stattfinden. Der Inhaberschaftstransfer kann implizit sein. Wenn beispielsweise der TCP-Spoofing-Kernel **280** eine TSK-Nachricht an den Backbone Protocol Kernel **282** für eine Übertragung über eine Backbone-Verbindung sendet, übergibt der TSK **280** die Inhaberschaft am Puffer implizit an den BPK **282**. In einem Ausführungsbeispiel darf nur der implizite Inhaber eines Puffers auf den Puffer zugreifen. Abgesehen von dem Fall, in dem der spezifische Kontext einer Schnittstelle so definiert ist, dass er es zulässt, sollte ein Kernel nicht annehmen, dass Felder in einem Puffer nicht verändert wurden, falls der Kernel einen Puffer außerhalb seines eigenen Kontexts weitergibt und diesen dann zurückbekommt.

[0139] Fig. 15 zeigt ein Diagramm eines IP-Pakets, das in dem System von Fig. 1 verwendet wird. Ein IP-Paket **1500** weist einen IP-Header **1501** (wie in EETF RFC **791** definiert, das hierin in seiner Gesamtheit durch Bezugnahme aufgenommen ist), gefolgt von einem Nutzinhalt **1503** auf. Der IP-Header **1501** weist im Allgemeinen eine Länge von 20 Byte auf. Der IP-Header **1501** kann länger als 20 Byte sein, wenn IP-Header-Optionen verwendet werden.

[0140] Die Größe des IP-Paket-Nutzinhalts **1503** wird durch die Größe der Maximum Transmission Unit (MTU) des Netzwerks, das verwendet wird, um das IP-Paket zu befördern, bestimmt. Beispielsweise ist die MTU einer Ethernet-Verknüpfung **1500** Byte, was einen IP-Paketnutzinhalt **1503** von bis zu 1480 Byte unterstützt. Wie in Fig. 15 dargestellt, trägt die IP-Paketnutzlast im Allgemeinen eine „Nachrichteneinheit" eines Protokolls einer höheren Schicht. Diese Protokolle höherer Schichten können User Datagram Protocol (UDP) **1505**, TCP **1507** und das PEP-Merkmal PBP **1509** einschließen. UDP **1505** schließt einen UDP-Header **1511** und einen Nutzinhalt **1513** (der die Daten enthält) ein. Auf ähnliche Weise stellt der TCP **1507** einen TCP-Header **1515** und einen Datenabschnitt **1517** bereit. In dem PBP 1509-Format sind beispielsweise eine TSK-Nachricht **1518** mit einem TSK-Header **1519** und Daten **1521** untergebracht. Die TSK-Nachricht **1518** stellt ihrerseits den Nutzinhalt, oder die Daten, **1523** eines PBP-Segments dar. Das PBP **1509** schließt auch einen Header **1525** ein.

[0141] Die Puffer werden zwischen der Umgebung **210** und den PEP-Kernels als IP-Pakete weitergegeben. An der TSK/BPK-Schnittstelle werden Puffer zwischen TSK **280** und BPK **282** als TSK-Nachrichten weitergegeben. Der gemeinsame PEP-Puffer-Header **1403**, der nachstehend ausführlicher beschrieben ist, wird verwendet, um die geeignete Puffer-Nutzlast an jede Schnittstelle weiterzugeben.

[0142] Fig. 16 zeigt ein Diagramm eines Formats des gemeinsamen PEP-Puffer-Headers gemäß einer Ausführungsform der vorliegenden Erfindung. Der gemeinsame Puffer-Header **1403** hat drei Zwecke: (1) um einen Mechanismus zum Weitergeben von Puffern zwischen einer Umgebung **210** und den PEP-Kernels und zwischen den verschiedenen

PEP-Kernels selbst bereitzustellen; (2) um einen Mechanismus bereitzustellen, der die Fähigkeit für IP, TCP, PBP und TSK-Header, zu wachsen (und zu schrumpfen), ohne eine Verschiebung der Daten in einem IP-Paket zu benötigen, unterstützt, wodurch die Leistung durch Vermeiden von Datenkopien deutlich verbessert wird; und (3) um Platz für inhaberspezifische Pufferfelder zu schaffen (die Notwendigkeit, separate Pufferdatenstrukturen zuzuordnen, abzuschaffen). Es sei darauf hingewiesen, dass die Grenze zwischen dem inhaberspezifischen „Header" und dem Header-Wachstums-„Header" etwas willkürlich ist, da der Kernel, falls erforderlich, inhaberspezifische Felder in den Header-Wachstumsplatz (und umgekehrt) eingeben kann, falls sie passen. Dies kann jedoch nur innerhalb eines Kernels passieren. Die Grenze zwischen den beiden „Headers" muss von einem Kernel respektiert werden, wenn er einen Puffer zur Umgebung oder zu einem anderen Kernel weitergibt.

[0143] Der allgemeine PEP-Puffer-Header **1403** schließt ein Flags+Offset-Feld **1601** ein, das (beispielsweise) 2 Byte lang ist, wodurch 4 Bit einem Flags-Feld **1601a** zugeordnet sind und die restlichen 12 Bit für das Nutzinhalt-Offset-Feld **1601** bereitgestellt sind. Was das Flags-Feld **1601a** betrifft, so hält das erste (bedeutendste Bit) Flag-Bit das Richtungs-(DIR-) Flag. Das Richtungs-Flag zeigt an, ob dieser spezielle Puffer in der LAN-zu-WAN-Richtung (DIR = 0) oder der WAN-zu-LAN-Richtung (DIR = 1) zugeordnet wurde. Das letzte (unbedeutendste Bit) Flag-Bit ist für die Verwendung durch die Plattformumgebung **210** reserviert. Die beiden mittleren Flag-Bits sind reserviert. Was das Nutzinhalt-Offset-Feld **1601b** betrifft, so spezifiziert das Feld **1601b** in Bytes den aktuellen Stand des Puffer-Nutzinhalts (z.B. des IP-Pakets). Der Header-Wachstumsplatz im Puffer ermöglicht es, diesen Wert sowohl nach oben als auch nach unten anzupassen. Es muss jedoch darauf geachtet werden, den Nutzinhalt-Offset bzw. -Versatz nicht über die Grenze zwischen dem inhaberspezifischen Feld **1605** und dem Header-Wachstumsfeld **1607** hinaus zu verändern.

[0144] Das Verbindungs-Handle-Feld **1603**, das 2 Byte lang ist, spezifiziert den Handle der Backbone-Verbindung, dem dieser Puffer zugeordnet wurde. Der Verbindungs-Handle kann beispielsweise in Puffern, die keine bandbreitenbedarfsreduzierten TCP-Segmente oder PBP-Segmente enthalten, und in Puffern, für die die Plattformumgebung **210** noch nicht die richtige Backbone-Verbindung, der der Puffer zugeordnet werden soll, bestimmt hat, auf 0 × FFFF eingestellt werden. Letzteres trifft auf TCP <SYN>-Segmente zu, die vom lokalen LAN empfangen werden.

[0145] Das 24 Byte große inhaberspezifische „Header"-Feld **1605** sorgt für die Verschiebung des Inhalts eines Puffers, um unterschiedliche Header-Größen aufnehmen zu können, die die CPU benötigt. Falls der Nutzinhalt des Puffers klein ist, muss die benötigte CPU nicht bedeutend sein. Wenn jedoch der Nutzinhalt groß ist, z.B. wenn der Puffer Anwenderdaten enthält, kann die benötigte CPU sehr bedeutend sein. Und da das Befördern von Anwenderdaten der eigentliche Zweck eines Netzwerks ist (und idealerweise den überwiegenden Teil des Verkehrs ausmacht) ist eine Optimierung für den Fall von großen Anwenderdatennachrichten wünschenswert. Die Größe eines empfangenen IP-Headers **1501** und die eines übertragenen IP-Headers **1051** sind im Allgemeinen gleich, d.h. 20 Byte. Daher erfordert der Austausch eines IP-Headers **1501** mit einem Anderen in der Regel kein spezielles Puffer-Management. Dagegen unterscheidet sich die Größe eines TCP-Headers **1515** von der Größe eines PBP-Header **1525** und sogar von der Größe von kombinierten PBP- und TSK-Headers. Ein TCP-Header **1515** ist im Algemeinen 20 Byte groß. Die Verwendung von TCP-Optionen kann die Größe des TCP-Headers **1515** vergrößern. Wie derzeit definiert, hat ein PBP-Header **1525** 12 Byte, wenn das PBP-Segment einen TSK-Nachricht einschließt. In den meisten Fällen hat ein TSK-Header **1519** (Fig. 15) für eine Datennachricht 6 Byte. Für die Ausnahmen ist der TSK-Header **1519** 18 Byte groß. Daher sind die kombinierten PBP- und TSK-Headers **1525** und **1519** für eine Datennachricht meistens 18 Byte groß.

[0146] An der Oberfläche kann es so aussehen, als ob das Austauschen entweder des PBP-Headers **1525** oder des TSK-Headers **1519**, so dass die kombinierten Headers **20** Bytes haben, um der Größe des TCP-Headers **1515** angepasst zu sein, die Puffer-Managementleistung verbessert (um den Preis der Verschwendung einiger Bytes im Overhead, wenn PBP-Segmente durch das WAN verschickt werden). Zusätzlich zur Verringerung der Flexibilität in Bezug auf den Umgang mit TCP-Optionen zeigt sich jedoch, dass dies nicht der Fall ist. Der Grund dafür ist, dass TSK- und BPK-Puffer-Management unabhängig voneinander stattfinden. Der TSK **280** kennt die Größe des PBP-Headers **1525** nicht und sollte sie auch nicht kennen. Und andererseits kennt der BPK **282** nicht die Größe des TSK-Headers und sollte sie auch nicht kennen. Wenn die Kernel die Header-Größe des jeweils anderen kennen würden, würde dies ihre Protokollschichtbeziehung stören und eine unbeabsichtigte Abhängigkeit zwischen den Kernels erzeugen. Das Verfahren, mit diesem Problem umzugehen, ist die Verwendung von extra Platz im vorderen Teil des Puffers zusammen mit einem „Zeiger" (d.h. einem Verschiebezähler) für den Puffernutzinhalt (d.h. den aktuellen Anfang des IP-Pakets). Dieses Verfahren macht es möglich, dass die Daten an Ort und Stelle bleiben und nur die Puffer-Header umherbewegt werden. Und es zieht Vorteil aus der Tatsache, dass die PEP-Kernels im Allge-

meinen den Platz für die Header nur neu verwenden. Felder in einem Header bleiben kaum unverändert und daher erfordert eine Verschiebung des Orts eines Headers einfach eine Änderung des Orts, an dem ein Kernel Felder füllen muss, nicht eine tatsächliche Verschiebung des Header-Inhalts. Beispielsweise enthält der IP-Header **1501**, der vom TCP-Spoofing-Kernel **280** benötigt wird, um TCP-Daten an den lokalen Host zu schicken und von diesem zu empfangen, keine Feldwerte, die er mit dem IP-Header **1501**, den der Backbone Protocol Kernel **282** benötigt, um die gleichen Daten durch das WAN zu senden und zu empfangen, gemeinsam hat. Und der TCP-Header **1515**, der verwendet wurde, um Daten zum lokalen Host zu schicken und von diesem zu empfangen, wird vollständig durch die PBP- und TSK-Header **1519** und **1509** ersetzt, die verwendet werden, um die gleichen Daten über das WAN zu senden und zu empfangen (und umgekehrt). In einem Ausführungsbeispiel kann in einem Puffer, der noch keine Header-Anpassungen erfahren hat, der Nutzinhalt-Offset am Anfang eines IP-Pakets 44 Byte in den Puffer hinein zeigen (weil der Puffer in diesem Beispiel mit 16 Byte Header-Wachstumsplatz initialisiert wird). Falls ein Header eingesetzt werden muss, der kleiner ist als der Header, den er ersetzt, dann verschiebt der Kernel, der die Anpassung vornimmt, die Header nach rechts, wodurch das Nutzinhaltfeld im Puffer aktualisiert wird. Falls ein Header eingesetzt werden muss, der größer ist als der Header, den er ersetzt, dann verschiebt der Kernel, der die Anpassung vornimmt, den Header nach links, wodurch wiederum das Nutzinhalt-Offset-Feld **1601b** im Puffer aktualisiert wird. Natürlich können, wie oben angegeben, selbst dann, wenn keine Header-Anpassungen erforderlich sind, Nutzinhalt-Offset-Anpassungen erforderlich sein, weil IP-Pakete nicht die Puffer-„Einheit" sind, die an allen Schnittstellen weitergegeben werden. Genauer sind TSK-Nachrichten die Puffer-„Einheit", die zwischen dem TCP-Spoofing-Kernel **280** und dem Backbone Protocol Kernel **282** weitergegeben wird.

[0147] Die Fig. 17 bis Fig. 20 zeigen die Verwendung des Header-Wachstumsplatzes für TCP-Datensegmente gemäß einer Ausführungsform der vorliegenden Erfindung. In Fig. 17 wird ein Puffer, der ein TCP-Datensegment enthält, das er vom lokalen LAN empfangen hat, von der Plattformumgebung **210** zum TCP-Spoofing-Kernel **280** weitergegeben. Der TSK **280** entfernt den 20 Byte-IP-Header **1501** und den 20 Byte-TCP-Header **1515** und fügt einen 6 Byte-TSK-Header **1519** hinzu, wobei der Nutzinhalt-Offset **1601b** von 44 auf 78 aktualisiert wird (was den Größenunterschied zwischen dem ursprünglichen und dem neuen Header darstellt), und gibt dann den Puffer als TSK-Nachricht an den Backbone Protocol Kernel **282** weiter. Der BPK **282** fügt einen 20 Byte-IP-Header **1501** und einen 12 Byte-PBP-Header **1525** zu der TSK-Nachricht hinzu, wodurch der Nutzinhalt-Offset **1601b** von 78 auf 46 aktualisiert wird, und gibt den Puffer dann an die Plattformumgebung **210** für die Weitersendung an das WAN weiter.

[0148] Fig. 18 stellt den gleichen Pufferfluss für den Fall dar, dass der TSK **280** einen 12 Byte-Verbindungs-Header **1515** für das TCP-Datensegment zusätzlich zum TSK-Header **1519** einsetzen muss.

[0149] In Fig. 19 wird ein Puffer, der eine TSK-Datennachricht, die er vom WAN erhalten hat, enthält, von der Plattformumgebung **210** zum BPK **282** weitergegeben. Der BPK **282** entfernt den 20 Byte-IP-Header **1501** und den 12 Byte-PBP-Header **1525**, wobei er den Nutzinhalt-Offset **1601b** von 44 auf 76 aktualisiert, und gibt dann den Puffer an den TSK **280** weiter. Der TSK **280** entfernt den 6 Byte-TSK-Header **1519** und fügt einen 20 Byte-IP-Header **1501** und einen 20 Byte-TCP-Header **1515** hinzu, um die TSK-Datennachricht in ein TCP-Datensegment umzuwandeln, wodurch der Nutzinhalt-Offset **1601b** von 76 auf 46 aktualisiert wird, und gibt dann den Puffer an die Plattformumgebung **210** für die Weitersendung an das WAN weiter.

[0150] Fig. 20 stellt den gleichen Pufferfluss für den Fall dar, dass der TSK **280** zusätzlich zum TSK-Header **1519** auch einen 12 Byte-TCP-Verbindungs-Header **1515** von der TSK-Datennachricht entfernen muss.

[0151] Eine Anfangsgröße von 16 Byte kann ausgewählt werden, da ein 16 Byte-Header-Wachstums-„Header" 4 Byte für eine Ausrichtung bereitstellt und eine Reserve für nicht vorhergesehene Header-Wachstumsanforderungen bereitstellt. In einer bestimmten Plattform kann die Plattformumgebung **210** jedoch so gewählt werden, dass sie eine Anfangs-Header-Wachstumsplatzgröße von über 16 Byte nutzt. Dies kann beispielsweise erwünscht sein, um Platz für einen Ethernet-MAC-Header zu schaffen, wodurch möglicherweise die Nutzung eines allgemeinen physischen Puffer-Pools, der von allen logischen Puffer-Pools gemeinsam verwendet wird, möglich ist.

[0152] Es sei darauf hingewiesen, dass nicht alle TCP-Segmente, die von dem TCP-Spoofing-Kernel verschickt werden, von TSK-Nachrichten stammen, die von einem TSK-Peer empfangen werden. Der TSK muss häufig „ganz von Grund auf" ein TCP-Segment (z.B. eine Bestätigung für ein empfangenes TCP-Datensegment) erzeugen, um es zu einem lokalen Host zu senden. Wie bereits angegeben, ruft der TSK **280**, wenn er solch ein TCP-Segment erzeugen muss, die Plattformumgebung **210** auf, einen WAN-zu-LAN-Puffer **1203** zuzuordnen. Der Puffer **1203**, der von der Umgebung **210** bereitgestellt wird, wird initialisiert, wobei sein Nutzinhalt-Offset **1601b** auf das erste Byte jenseits des Default-Hea-

der-Wachstums-„Header" der Plattform zeigt (z.B. 44 Byte in den Puffer hinein). Da keine Header vor den Headern eingesetzt werden müssen, die vom TSK **280** eingesetzt werden (abgesehen von dem LAN MAC-Header, der für alle IP-Pakete eingesetzt wird), muss der TSK **280** sich nicht damit befassen, Platz für zusätzliche Header im Puffer frei zu lassen. Der TSK **280** kann einen IP-Header **1501** und einen TCP-Header **1515** an dem Ort, der von der Plattformumgebung **210** bereitgestellt wird, einsetzen. Dies ist in Fig. 21 dargestellt.

[0153] Ebenso stammen nicht alle PBP-Segmente, die vom Backbone Protocol Kernel **282** verschickt werden, von TSK-Nachrichten, die vom TSK **280** weitergeschickt werden. Der BPK **282** muss häufig „von Grund auf" ein PBP-Segment (z.B. eine Bestätigung für ein empfangenes PBP-Datensegment) erzeugen, um es an einen BPK-Peer zu schicken. Wenn der BPK **282** ein solches PBP-Segment erzeugen muss, ruft der BPK **282** die Plattformumgebung **210** auf, um einen Lan-zu-Wan-Puffer **1201** zuzuordnen. Der von der Umgebung bereitgestellte Puffer wird initialisiert, wobei sein Nutzinhalt-Offset auf das erste Byte jenseits des Default-Header-Wachstums-„Header" der Plattform zeigt (z.B. 44 Byte in den Puffer). Da keine Header vor den vom BPK **282** eingesetzten Headern eingesetzt werden müssen (abgesehen von einem etwaigen WAN MAC-Header, der für alle IP-Pakete eingesetzt wird), muss sich der BPK **282** nicht damit aufhalten, Platz für zusätzliche Header im Puffer frei zu lassen, und kann einen IP-Header **1501** und einen PBP-Header **1525** an dem Ort einsetzen, der von der Umgebung **210** bereitgestellt wird. Dies ist in Fig. 22 dargestellt.

[0154] Der BPK **282** muss niemals Nachrichten für einen lokalen Host (über TSK **280**) erzeugen. Jedoch muss der TSK **280** TSK-Nachrichten (z.B. eine Connection Terminated-Nachricht, wenn eine Verbindung aufgrund von Rückübermittlungsfehlern beendet wird) erzeugen, um sie zu TSK-Peers zu schicken (über BPK **282**). Wenn der TSK **280** eine TSK-Nachricht erzeugen muss, ruft der TSK **280** die Plattformumgebung **210** auf, einen LAN-zu-Wan-Puffer **1201** zuzuordnen. Wie in den anderen oben beschriebenen Fällen wird der von der Umgebung bereitgestellte Puffer initialisiert, wobei sein Nutzinhalt-Offset **1601b** auf das erste Byte jenseits des Default-Header-Wachstums-„Header" der Plattform zeigt (z.B. 44 Byte in den Puffer). Da eine TSK-Nachricht über BPK **282** verschickt wird, muss der TSK **280** in diesem Fall jedoch Platz für den BPK **282** lassen, um PBP- und IP-Header einzusetzen; dies erfordert jedoch nicht, dass der TSK **280** etwas über die Größe des PBP-Headers **1525** weiß. Der TSK **280** kann einfach den TSK-Header **1519** (und, falls nötig, den TCP-Verbindungs-Header **1515**) an den Orten hinzufügen, wo er dies getan hätte, wenn der Puffer mit einem IP-Header **1501** und einem TCP-Header **1515** darin empfangen worden wäre, wie in Fig. 23 dargestellt.

[0155] Es sei darauf hingewiesen, dass das Szenario in Fig. 23, wodurch TSK **280** den Puffer so einstellt, dass ein IP-Header **1501** und ein TCP-Header **1515** eingeschlossen werden, so dass der Puffer gleich aussieht als wäre der Puffer von dem lokalen LAN empfangen worden, nur der Erläuterung dient. Diese Implementierung kann beispielsweise sofort den TSK-Header **1519** an der richtigen Stelle einwechseln, indem der richtige Wert zum Nutzinhalt-Offset **1601b** addiert wird. Wenn der TSK **280** und der BPK **282** die Umgebung aufrufen, einen Puffer zuzuordnen, stellen sie die Größe des Puffers bereit, den sie zuordnen wollen. Die angegebene Größe reflektiert jedoch nur die Größe des Segments oder der Nachricht, die sie erzeugen wollen (einschließlich der relevanten Protokoll-Header). Die Größe schließt nicht den PEP-Puffer-Header oder einen etwaig benötigten plattformspezifischen Puffer-Header ein. Die Umgebung **210** passt die angeforderte Größe dementsprechend an. Diese Anpassung wird aus zwei Gründen der Umgebung überlassen. Zunächst wissen der TSK **280** und der BPK **282** nichts über plattformspezifische Puffer-Header. Zweitens könnte eine bestimmte Umgebung **210** einen größeren Header-Wachstumsplatz als das erforderliche Minimum nutzen wollen.

[0156] Es gibt ein weiteres Puffernutzungsszenario, das den TCP-Spoofing-Kernel einschließt, wobei eine „Nachricht" empfangen und nicht weitergesendet wird, aber der Empfang der „Nachricht" es erfordert, dass eine andere „Nachricht" in der gleichen Richtung (LAN-zu-WAN oder WAN-zu-LAN) wie die ursprüngliche „Nachricht" erzeugt wird. Zum Beispiel wird ein empfangenes TCP <RST>-Segment nicht weitergesendet, kann aber die Notwendigkeit bewirken, eine Connection Terminated-TSK-Nachricht zu senden, und umgekehrt. Wenn dies der Fall ist, nutzt TSK **280** erneut den Puffer der ursprünglichen „Nachricht", um die neue „Nachricht" zu erzeugen, anstatt den empfangenen Puffer freizugeben und einen neuen zuzuordnen. Dies erfordert jedoch keinerlei spezielle Handhabung aufgrund der Tatsache, dass der TSK **280** die Header von TCP-Segmenten und TSK-Nachrichten, die er empfängt, vor dem Weitersenden bereits vollständig austauscht. Die gleichen Puffernutzinhalt-Offset-Anpassungen, die für weitergeschickte Daten-„Nachrichten" durchgeführt werden, funktionieren auch bei der erneuten Verwendung eines Puffers. Dies ist in den Fig. 24 und Fig. 25 dargestellt, die zeigen, dass die gleiche Header-Anpassung, die in den Fig. 18 und Fig. 20 dargestellt ist, verwendet werden kann; der einzige Unterschied ist, dass keine Daten in dem Puffer sind, die für den Wiederverwendungsfall behalten werden müssen.

[0157] Wie bereits angegeben, spezifizieren der TSK **280** oder der BPK **282** die benötigte Puffergröße, wenn sie einen Puffer zuweisen, um ein zu verschickendes IP-Paket zu konstruieren. Der BPK **282** erzeugt keine IP-Pakete, die in Richtung des lokalen LAN weitergeleitet werden müssen (mittels des TSK **280**), und daher ist er nicht damit befasst, Platz für den TSK **280** zu lassen, um Daten in den zugeordneten Puffer einzusetzen. Jedoch erzeugt der TSK **280** TSK-Nachrichten, die zum WAN weitergeleitet werden sollen (mittels des BPK **282**). Wenn der TSK **280** einen Puffer für den Zweck des Verschickens einer Nachricht zuordnet, muss der TSK **280** daher Platz für den PBP-Header **1525** lassen. Jedoch setzt der BPK **282** den PBP-Header **1525** vor dem TSK-Header **1519** ein, wobei er die TSK-Nachricht wie Daten behandelt.

[0158] Solange ein TSK **280** der obigen Strategie des „Einsetzens" von Platz für den IP-Header **1501** und den TCP-Header **1415** folgt, bleibt die Größe des zugeordneten Puffers die richtige. Die Größe eines Puffers kann jedoch nicht mehr die richtige sein, wenn er erneut verwendet wird. Beispielsweise wird ein empfangenes TCP <SYN>-Segment in der Regel ein 40 Byte-IP-Paket sein. Aber das IP-Paket, das für die TSK CR-Nachricht verwendet wird, die als Folge des Empfangs des TCP <SYN>-Segments verschickt werden muss, wird größer sein als 40 Byte. Falls eine Strategie für eine variable, exakte Puffergröße in der PEP-Endpunktplattform **210** verwendet wird, bleibt kein Platz im Puffer, um die CR-Nachricht zu erstellen. Es gibt zwei Möglichkeiten, dieses Problem zu lösen. Die erste Möglichkeit ist die Nicht-Zulassung der Wiederverwendung eines Puffers in diesem Fall. Der TSK **280** könnte gezwungen werden, den ursprünglichen Puffer freizugeben und einen neuen, größeren Puffer zuzuordnen. Die zweite Möglichkeit ist, eine Plattformumgebung **210** dazu zu bringen, immer einen Puffer von mindestens einer gewissen Mindestgröße zuzuordnen, wenn ein Puffer erforderlich ist oder wenn die Umgebung **210** ein empfangenes TCP- oder PBP-Segment aus einem LAN Rx-Puffer **1205** oder einem WAN Rx-Puffer **1209** in einen LAN-zu-WAN-Puffer **1201** oder einen WAN-zu-LAN-Puffer **1203** kopiert. Dies ist der Ansatz, der den PEP-Kernel-Code vorteilhafterweise vereinfacht.

[0159] Selbst wenn eine Plattform eine Festgrößenpuffer-Strategie verwendet, besteht trotzdem die Notwendigkeit, eine minimale Puffergröße zu erzwingen. In diesem Fall ist die minimale Puffergröße erforderlich, um sicherzustellen, dass alle Felder, auf die von einem PEP-Kernel zugegriffen wird, sich im ersten Puffer befinden, der ein IP-Paket enthält.

[0160] Dies schließt alle Protokoll-Header und die Daten für TSK-Steuernachrichten ein. Dies trifft zu, wenn die Pufferstrategie darin besteht, einzelne Festgrößenpuffer zu verwenden, da dies die Verwendung großer Puffer erfordert. Falls jedoch eine Speicherverkettung angewendet wird, dann muss der erste Puffer in der Kette groß genug sein, um alle Informationen aufzunehmen, auf die von den PEP-Kernels zugegriffen werden muss. Beispielsweise kann die minimale Puffergröße als 100 Byte festgelegt sein; d.h. eine Plattformumgebung **210** darf keinen Puffer zuordnen, der kleiner ist als 100 Byte. Der minimale Wert kann konfiguriert werden. Eine Plattformumgebung **210** kann auch eine minimale Puffergröße von mehr als 100 Byte verwenden, falls gewünscht; beispielsweise um die Pufferausrichtungsleistung zu verbessern. Die Erzwingung einer minimalen Puffergröße ist die Aufgabe der Plattformumgebung **210**. Die Tatsache, dass der Puffer, der von der Umgebung zurückgeschickt wird, größer sein kann als angefordert, ist für den TSK **280** und BPK **282** erkennbar.

[0161] Die verschiedenen PEP-Kernels müssen häufig Pufferreihen miteinander verketten, um Warteschlangen zu implementieren. Dies kann durch Zuordnen eines kleinen, separaten Pufferdeskriptorblocks und anschließende Verwendung von Feldern im Pufferdeskriptor, um auf einen Puffer zu zeigen und um Pufferdeskriptoren miteinander zu verknüpfen, implementiert werden. Da jedoch im Grunde eine eins-zu-eins-Beziehung zwischen der Zahl der erforderlichen Pufferdeskriptoren und der Zahl der erforderlichen Puffer besteht, ist ein alternativer Ansatz die grundsätzliche Einbettung des Pufferdeskriptors in den Puffer selbst. Dies ist der Zweck des inhaberspezifischen Teils **1605** des allgemeinen PEP-Puffer-Headers **1403**. Der inhaberspezifische „Header" **1605** steht dem aktuellen Inhaber eines Puffers zur Verfügung, um mit einer kernelspezifischen Pufferdeskriptorarchitektur überschrieben zu werden. Der Inhaberkernel kann dann den Pufferdeskriptor verwenden, um Puffer miteinander zu verknüpfen. Darüber hinaus (oder sogar als Alternative) kann der Inhaberkernel den inhaberspezifischen „Header" nutzen, um kernelspezifische Informationen, die mit dem Puffer in Beziehung stehen (beispielsweise einen Zeitnehmer, der mit dem Puffer assoziiert ist) zu speichern. Fig. 26 zeigt ein Beispiel dafür, wie ein Kernel den inhaberspezifischen „Header" **1403** verwenden könnte. Wie bereits erörtert, gibt ein Kernel die implizite Inhaberschaft an einem Puffer auf, wenn er diesen an die Umgebung **210** oder einen anderen Kernel weitergibt. Daher sollte ein Kernel nicht annehmen, dass irgendwelche Felder in Sätzen im inhaberspezifischen Teil des allgemeinen PEP-Puffer-Headers **1403** sich in einem Puffer, der diesen abgibt und dann zurückbekommt, nicht verändern, solange die Semantik der speziellen Schnittstelle spezifisch so definiert ist, dass sie diese Annahme zulässt. Wenn der Backbone Protocol Kernel **282** beispielsweise ein PBP-Segment an die Plattformumgebung **210** für die Übertragung weitergibt, sollte der BPK **282** nicht an-

nehmen, dass irgendwelche Felder, die er im inhaberspezifischen „Header" definiert hat, sich nicht verändert haben, wenn er den Puffer zurückbekommt, solange die Definition der spezifischen prozeduralen Schnittstelle nicht angibt, dass diese Annahme gültig ist.

[0162] Da sowieso eine Kopie erforderlich ist (aufgrund der Art, wie die verschiedenen LAN- und WAN-„Treiber" arbeiten), nutzt ein IP-Gateway (d.h. ein PEP-Endpunkt **210**) die vorhandene Implementierung für den LAN Rx-Puffer **1205**, den LAN Tx-Puffer **1207**, den WAN Rx-Puffer **1209** und den WAN Tx-Puffer **1211**. Ein einziger physischer Speicher-Pool wird sowohl für die LAN-zu-WAN- als auch für die WAN-zu-LAN-Puffer-Pools **1201** und **1203** verwendet. Das IP-Gateway **210** kann den Ansatz eines einzelnen Puffers variabler Größe zum Zuordnen von PEP-Puffern anwenden. Einzelner Puffer bezieht sich auf die Tatsache, dass nur ein physischer Puffer erforderlich ist, um ein ganzes IP-Paket aufzunehmen. Variable Größe bezieht sich auf die Tatsache, dass die Größe des zugeordneten Puffers (abgesehen von der Einschränkung einer minimalen Puffergröße, wie oben beschrieben) für die Größe des IP-Pakets exakt passen wird (wobei Platz für die verschiedenen Puffer-Header bleibt). Die malloc()- und free()-Funktionen verfolgen die exakte Größe der Puffer. Daher muss die IP-Gateway-Implementierung des PEP-Endpunkts **210** keinen plattformspezifischen Puffer-Header benötigen.

[0163] Im Hinblick auf die anderen PEP-Endpunkt-Implementierungen sind das Multimedia VSAT-Puffermanagement und das Multimedia Relay-Puffermangagement dem IP-Gateway-Puffermanagement ähnlich. Genauer implementieren die VSATs ihren LAN-zu-WAN-Pufferpool **1203** und ihren WAN-zu-LAN-Pufferpool **1203** als Pools von Speichern mit Puffern, die anhand von malloc() zugeordnet und anhand von free() freigegeben werden. Ein einziger physischer Speicherpool wird sowohl für den LAN-zu-WAN-Puffer **1201** als auch für den WAN-zu-LAN-Pufferpool **1203** verwendet. Ein Ansatz für einzelne Puffer variabler Größe zum Zuordnen von PEP-Puffern wird angewendet. Anders als beim IP-Gateway-Ansatz schließen die VSATs jedoch einen plattformspezifischen Puffer-Header in jedem Puffer ein.

[0164] Was die PES Remote PEP-Plattformumgebung **210** betrifft, so ist die Verwendung von Verkettungen kleiner Puffer erforderlich, um IP-Pakete aufzunehmen. Um die Tatsache, dass verkettete kleine Puffer verwendet werden, vor den PEP-Kernels zu verbergen, muss die PES Remote-Plattformumgebung **210** sicherstellen, dass alle Header, einschließlich des allgemeinen PEP-Puffer-Headers, in den ersten Puffer einer Pufferkette passen, und nicht nur die Protokoll-Header. Um diese Anforderung zu erfüllen, führt die PES Remote-Umgebung folgendes für ein IP-Paket durch, das von dem lokalen LAN oder dem WAN empfangen wird. Falls die Länge (oder der Inhalt) des ersten Puffers in der Kette klein genug ist, damit der allgemeine PEP-Puffer-Header **1403** in den Puffer eingesetzt werden kann, wird der Inhalt des Puffers nach rechts verschoben, um Platz dafür zu machen. Im Allgemeinen werden Pufferketten empfangen, deren Puffer alle voll sind, abgesehen vom letzten Puffer. Daher werden diese Bedingungen, wiederum im Allgemeinen, nur dann erfüllt, wenn das gesamte IP-Paket in einen einzigen kleinen Puffer passt. Diese Möglichkeit ist in Fig. 27 dargestellt. Falls die Länge des ersten Puffers zu groß ist, um zu ermöglichen, dass der gemeinsame PEP-Puffer-Header eingesetzt wird, ordnet die Umgebung **210** einen extra Puffer zu und stellt ihn der Pufferkette voran. Falls kein Puffer verfügbar ist, wird das IP-Paket fallen gelassen und muss wiederhergestellt werden als wäre es beim Durchgang durch das LAN oder WAN verloren gegangen. Der gemeinsame PEP-Puffer-Header wird in den extra Puffer eingegeben und dann werden alle Protokoll-Header im ursprünglichen ersten Puffer in den Puffer kopiert. Schließlich werden jegliche Daten, die im ursprünglichen ersten Puffer zurückgeblieben sind, nach links verschoben (nach vorne im Puffer). Diese Möglichkeit ist in Fig. 28 dargestellt. Obwohl diese Kopien zusätzlichen Platzbedarf darstellen, ist eine gewisse Art von Kopieren unabdingbar und dieser Ansatz sollte den Umfang des Kopierens auf ein Minimum beschränken. Darüber hinaus können im PES Remote für den LAN Rx-Pufferpool **1205**, den LAN-zu-WAN-Pufferpool **1201** und den (Inroute) WAN Tx-Pufferpool **1211** tatsächlich die gleichen Puffer verwendet werden. Und die gleichen Puffer **1201**, **1205** und **1211** können auch für den (Outroute) WAN Rx-Pufferpool **1209**, den WAN-zu-LAN-Pufferpool **1203** und den LAN Tx-Pufferpool **1207** verwendet werden. Somit kann die PES Remote-Plattformumgebung **210** beispielsweise vermeiden, dass einige der Kopien, die in anderen Arten von PEP-Endpunkt-Plattformen **210** erforderlich sind, Daten von einer Pufferart zu einer anderen bewegen, wodurch der CPU-Nachteil, der von den oben beschriebenen Kopien auferlegt wurde, wegfällt. In einem PES Remote kann die Größe eines LAN Rx-Puffers **1205**, eines LAN-zu-WAN-Puffers **1201** und eines (Inroute) WAN Tx-Puffers **1211** entweder 146 Byte oder 246 Byte betragen (je nach dem speziellen Software-Bau). Die Größe anderer Arten von Puffern beträgt 246 Byte. Auch mit dem gemeinsamen PEP-Puffer-Header **1403** stellen 146 Byte ausreichend Platz für die meisten IP-Pakete (z.B. TCP-Bestätigungen) und für viele Daten-IP-Pakete (z.B. HTTP GETs) bereit. Insbesondere stellen 146 Byte ausreichend Platz bereit, um jedes Segment oder jede Nachricht unterzubringen, die vom TSK **280** oder vom BPK **282** (von Grund auf) erzeugt werden muss.

[0165] Die Plattformumgebung **210** verfolgt die Menge des Pufferplatzes, der in jeder Richtung für jede Backbone-Verbindung verwendet wird. Diese Verfolgung wird für die Zwecke der Aufteilung von Pufferplatzressourcen in Hinblick auf Bekanntmachungen von TCP- und PBP-Fenstern durchgeführt. Zumindest in der Anfangsausgabe des PEP-Merkmals stützt die Umgebung **210** Entscheidungen im Hinblick auf die Zuordnung eines Puffers nicht auf die Menge an verwendetem Pufferplatz. Wenn sich die Notwendigkeit ergibt, einen Puffer zuzuordnen, ordnet die Umgebung **210** den Puffer zu, falls ein Puffer aus dem geeigneten Pufferpool verfügbar ist. Diese Vorgehensweise wirft keinerlei Probleme dahingehend auf, dass erwartet wird, dass TCP- und PBP-Sender (d.h. lokale TCP-Hosts und PEP-Endpunkt-Peers) keine Pakete über das hinaus befördern, was durch die bekannt gemachten Empfangsfenster, die sie von einem PEP-Endpunkt **210** erhalten, zugelassen ist. Diese Vorgehensweise vereinfacht das Fehlermanagement im Zusammenhang mit der Zuordnung von Puffern, um Steuernachrichten zu schicken, wenn der Pufferplatz ausgeht, erheblich. Die folgenden Abschnitte beschreiben die Verfolgung und Verwendung der Pufferplatzverfügbarkeit, um TCP- und PBP-Fenster zu berechnen.

[0166] Sowohl TCP als auch PBP verwenden Fenster, die vom Datenempfänger an den Datensender geschickt werden, um zu steuern, wie viele Daten vom Sender zum Empfänger übermittelt werden können. Generell ermöglicht ein größeres Fenster einen höheren Durchsatz. Der Durchsatz ist jedoch von der Größe der kleinsten Verknüpfung der Leitung, durch die die Daten fließen, beschränkt, deshalb erhöht jenseits eines bestimmten Punkts eine Vergrößerung des Fensters den Durchsatz nicht weiter. Um sicherzustellen, dass die übermittelten Daten vom Empfänger nicht verworfen werden, wenn sie ankommen, beschränkt der Empfänger im Allgemeinen das Fenster, das er bekannt gibt, aufgrund der Menge an Pufferplatz, die gegenwärtig zur Verfügung steht, um Daten zu empfangen. Um die Menge an verfügbarem Pufferplatz als Beschränkung der Fenstergröße verwenden zu können, muss der Empfänger jedoch wissen, wie viel Platz zur Verfügung steht. Um die Berechnung der Fenstergröße aufgrund des verfügbaren Pufferplatzes zu unterstützen, verfolgt die Plattformumgebung **210** die Menge an LAN-zu-WAN- und WAN-zu-LAN-Pufferplatz, der für jede Backbone-Verbindung verwendet wird (im Umgebungssteuerblock der Backbone-Verbindung (ECB)). Sobald die Umgebung **210** ein empfangenes TCP-Segment aus einem LAN Rx-Puffer **1205** in einen LAN-zu-WAN-Puffer **1201** kopiert, inkrementiert die Umgebung **210** die Menge an Pufferplatz, der für die Backbone-Verbindung verwendet wird, um den Bandbreitenbedarf der TCP-Verbindung, zu der das TCP-Segment gehört, zu reduzieren. Die Umgebung **210** bestimmt, welche Backbone-Verbindung gerade verwendet wird, indem sie in den CCB der TCP-Verbindung schaut. Für den Fall, dass ein TCP <SYN>-Segment empfangen wird, ohne dass bisher ein CCB für die TCP-Verbindung zugeordnet wurde, zählt die Umgebung **210** den Puffer des TCP <SYN>, wenn der TCP-Spoofing-Kernel **280** den CCB zuordnet. Die Umgebung **210** inkrementiert auch die LAN-zu-WAN-Pufferplatzzählung, sobald der TSK **280** einen LAN-zu-WAN-Puffer **1201** zuordnet, um eine TSK-Nachricht von Grund auf zu erzeugen, und sobald der Backbone Protocol Kernel **282** einen LAN-zu-WAN-Puffer **1201** zuordnet, um ein PBP-Segment von Grund auf zu erzeugen.

[0167] Die WAN-zu-LAN-Pufferbuchung funktioniert ähnlich wie die LAN-zu-WAN-Pufferbuchung. Wenn die Umgebung **210** ein empfangenes PBP-Segment von einem WAN Rx-Puffer **1209** in einen WAN-zu-Lan-Puffer **1203** kopiert, inkrementiert die Umgebung **210** die Menge an Pufferplatz, der für die Backbone-Verbindung verwendet wird, von der das PBP-Segment empfangen wurde. In einem Ausführungsbeispiel der Erfindung bestimmt die Umgebung **210** den Handle der Backbone-Verbindung durch Kombinieren des Peer-Index, der der Quell-IP-Adresse in dem IP-Paket zugeordnet ist, mit der Priorität der Verbindung (angezeigt durch di PBP-Portnummer). Die Umgebung **210** inkrementiert auch die WAN-zu-LAN-Pufferplatzzählung, wenn der TSK **280** einen WAN-zu-LAN-Puffer **1203** zuordnet, um ein TCP-Segment von Grund auf zu erzeugen. Die Umgebung dekrementiert die LAN-zu-WAN- oder WAN-zu-LAN-Pufferplatzzählung nach Bedarf, sobald ein Puffer freigegeben wird. Der Backbone-Verbindungs-Handle, der verwendet wird, um den geeigneten ECB zu finden, und ein LAN-zu-WAN versus WAN-zu-LAN-Flag werden in dem Puffer untergebracht, um die Buchung der des freigegebenen Puffers zu vereinfachen. Wie nachstehend beschrieben, wird Pufferplatz intern hinsichtlich der Nummer der verwendeten Puffer verfolgt, nicht anhand des verwendeten Pufferplatzes.

[0168] In einem Ausführungsbeispiel beeinflussen vier Arten von Parametern, die für einen PEP-Endpunkt **210** konfiguriert sind, die Verwendung von Pufferplatz, um die Bekanntgabe von Fenstergrößenwerte zu bestimmen: (1) Pufferplatz pro Peer, (2) TCP-Verbindungssteuerblöcke pro Peer, (3) Prozentanteil Ressourcen pro Verbindung und (4) Fenstergrößen-Obergrenze. Wie in Fig. 5 dargestellt, wird jeder PEP-Endpunkt **501**, **503** mit der Menge an Pufferplatz (angegeben in Kilobyte-Einheiten) konfiguriert (über sein PEP-Endpunktprofil), den er für WAN-zu-LAN-Verkehr verwenden sollte, der von jedem seiner PEP-Endpunkt-Peers empfangen wird. Dies ist die Gesamtmenge an WAN-zu-LAN-Pufferplatz in einem Fern-Site-PEP-Endpunkt **503** (der nur einen Peer hat). Dies ist der WAN-zu-LAN-Pufferplatz pro Peer in einem Hub-Site-PEP-Endpunkt **501**.

Wenn beispielsweise der Wert, der im PEP-Endpunktprofil des Hub-Site-PEP-Endpunkts konfiguriert wird, 500 kB beträgt, ist der WAN-zu-LAN-Pufferpool **1203** für jeden der Peers 500 kB. Falls **100** Peers vorhanden sind, dann ist die Gesamtmenge an WAN-zu-LAN-Pufferplatz 50 MB. Beim Konfigurieren des WAN-zu-LAN-Pufferplatzwerts muss der Operator die Gesamtmenge an verfügbarem Pufferplatz, die Zahl der Peers, die sich den Gesamtpool teilen müssen, und die Menge an Pufferplatz, die in LAN-zu-WAN-Richtung erforderlich ist, berücksichtigen. Die Menge an Pufferplatz, die in LAN-zu-WAN-Richtung erforderlich ist, ist nominal die Summe aller WAN-zu-LAN-Pufferplatzwerte aller PEP-Endpunkt-Peers. Der Operator kann jedoch tatsächlich den Pufferplatz überbuchen; d.h. der Operator ist nicht darauf beschränkt, die Menge an zu verwendendem Pufferplatz so zu konfigurieren, dass die Gesamtsumme, wenn alle Puffer verwendet würden, unter der tatsächlich verfügbaren Menge liegt. Der Operator könnte dies tun, um zu bewirken, dass größere Fenster bekannt gegeben werden, um den Durchsatz zu verbessern (jedoch auf Kosten des Risikos eines Paketverlusts) oder um sich Kenntnisse hinsichtlich dieser Anwendungen zunutze zu machen. Beispielsweise könnte der Operator wissen, dass diese Anwendungen am Tag mehr LAN-zu-WAN-Pufferplatz brauchen und in der Nacht mehr WAN-zu-LAN-Pufferplatz brauchen. Insbesondere wird der Operator den Pufferplatz im Hub-Site-PEP-Endpunkt **501** in der Regel überbuchen, weil es statistisch sehr unwahrscheinlich ist, dass Verkehr gleichzeitig zu jedem Peer geschickt wird. Pufferplatz wird in Byte vom Operator angegeben, da die Menge an Speicher (in Byte) das ist, was dem Operator bekannt ist.

[0169] Intern wandelt ein PEP-Endpunkt **501**, **503** den konfigurierten WAN-zu-LAN-Pufferplatz pro Peer aus der Byte-Zahl in die Pufferzahl um, um die Puffer zu verfolgen. Dies wird durch Teilen der Zahl der Bytes durch die Größe eines Puffers, der in der Lage ist, ein IP-Paket maximaler Größe unterzubringen (z.B. 1500 Byte plus die Größe des gemeinsamen PEP-Puffer-Headers plus die Größe eines etwaigen plattformspezifischen Headers) durchgeführt. Dies wird aus zwei Gründen getan. Erstens gibt der PBP Fenster ausgedrückt in Paketen, nicht in Byte, bekannt und muss annehmen, dass jedes Paket, das er empfängt, ein Paket maximaler Größe ist. Zweitens werden alle Pufferplatzberechnungen unter der Annahme eines IP-Pakets maximaler Größe gemacht, um Annahmen über die verwendete Pufferstrategie in einem PEP-Endpunkt-Peer zu eliminieren. Wenn beispielsweise ein PEP-Endpunkt eine variable Strategie eines Puffers mit exakter Größe anwendet und Bytes aufgrund von tatsächlichen IP-Paketgrößen zählt, aber sein Peer eine Festgrößen-Pufferstrategie anwendet, wird die Byte-Zählung die Menge an verwendetem Speicher in dem Peer nicht exakt wiedergegeben, solange nicht alle IP-Pakete die maximale Größe haben. Ebenso sorgt die Tatsache, dass die Menge an Pufferplatz überbucht werden kann, für ein großes Maß an Flexibilität im Hinblick auf die Leistungssteigerung. Und sie stellt einen Spielraum zum Kompensieren von Annahmen, die für ein bestimmtes Kundennetzwerk nicht zutreffen, bereit. Falls beispielsweise die maximale IP-Paketgröße in einem bestimmten Netzwerk 1000 Byte statt 1500 Byte ist und der Kunde nur PEP-Endpunktplattformen verwendet, die eine variable Exaktgrößen-Pufferstrategie anwenden, kann der Operator den WAN-zu-LAN-Pufferplatzparameter um 50 % erhöhen, um die Verwendung von IP-Paketen geringerer Größe zu kompensieren.

[0170] Die Zahl der TCP-Verbindungssteuerblöcke, die pro PEP-Endpunkt-Peer verwendet werden können, kann auch konfiguriert werden. Der Wert wird in erster Linie verwendet, um zu bestimmen, ob ein CCB in einem TSK 280-Peer verfügbar ist, um den Bandbreitenbedarf einer neu entdeckten TCP-Verbindung zu reduzieren. Dieser Wert beeinflusst jedoch auch Pufferplatzberechnungen im Zusammenhang mit Fenstergrößen, da der Pufferplatz, der für eine Backbone-Verbindung zur Verfügung steht, auf alle TCP-Verbindungen verteilt werden muss, die die Backbone-Verbindung nutzen. Die Pufferpoolgrößen-Berechnungen sind wie folgt:

$$S = S_O + 1500$$

$$n_b = N_b/S,$$

wobei S_O die Puffer-Overhead-Größe in Byte (z.B. gemeinsamer PEP-Puffer-Header und etwaiger plattformspezifischer Puffer-Header) ist, S die Puffergröße in Byte ist, N_b der konfigurierte Pufferplatz in Byte ist und n_b der Pufferplatz in einer Reihe von Puffern ist.

[0171] Mit Priorisierung bestehen möglicherweise mehrere Backbone-Verbindungen zwischen PEP-Endpunkt-Peers. Daher muss der Operator, zusätzlich zur Angabe der Menge an Pufferplatz (und der Zahl der CCBs) zur Verwendung für einen bestimmten PEP-Endpunkt-Peer, die Zuordnung dieser Ressourcen zu den verschiedenen Backbone-Verbindungen spezifizieren. Dies wird durch die Konfiguration von Ressourcen-Prozentanteilen, die jeder Priorität einer Backbone-Verbindung zugeschrieben werden, im Konnektivitätsprofil, das verwendet wird, um die Verbindungen zu definieren, durchgeführt. Für jede Priorität schreibt der Operator einen Ressourcen-Prozentanteil im Bereich von 0 5 bis 100 % zu (wobei 0 % verwendet wird, um anzuzeigen, dass bei dieser Priorität keine Backbone-Verbindung erforderlich ist). Der Operator kann Ressourcen durch Zuschreiben von Prozentanteilen, die sich auf mehr als 100 % summieren, überbuchen. Der Operator kann

Ressourcen durch Zuschreiben von Prozentanteilen, die sich auf unter 100 % summieren, auch unterbuchen; dies könnte von Nutzen sein, um Pufferplatz für die Verwendung von nicht-bandbreitenbedarfsreduziertem Verkehr (z.B. UDP) zu reservieren. Die Umgebung **210** verwendet die vom Operator konfigurierten Prozentanteile, wenn die Backbone-Verbindungen geöffnet werden. Die Menge an WAN-zu-LAN-Pufferplatz, der einer Backbone-Verbindung zugeschrieben wird, wird gleich dem WAN-zu-LAN-Pufferplatz pro Peer gesetzt, multipliziert mit dem Ressourcen-Prozentanteil, der dieser Backbone-Verbindung zugeschrieben ist. Ebenso wird die Zahl der CCBs, die mit dieser Backbone-Verbindung genutzt werden können, gleich der Zahl der CCBs pro Peer, multipliziert mit dem gleichen Ressourcen-Prozentanteil gesetzt. In einem Ausführungsbeispiel können Pufferplatz und CCBs unterschiedliche Prozentzahlen zugeschrieben werden; alternativ dazu kann ein einziger Parameter für beide verwendet werden. Die WAN-zu-LAN-Pufferplatz- und CCB-Grenzen-Berechnungen sind wie folgt:

$$B_{pi}^{W2L} = n_b \,\&\cdot X_{pi}$$

$$CCB_{pi} = CCB_e \,\&\cdot X_{pi},$$

wobei X_{pi} der Ressourcen-Prozentanteil für die Backbone-Verbindung zum Peer „p" bei Priorität „i" ist, B_{pi}^{WZL} die WAN-zu-LAN-Pufferplatzgrenze für die Backbone-Verbindung zum Peer „p" bei Priorität „i" ist, CCBpi die CCB-Grenze für die Backbone-Verbindung zum Peer „p" bei Priorität „i" ist und CCBe die konfigurierte PEP-Endpunkt-CCB-Grenze ist. Es sei darauf hingewiesen, dass die CCB-Grenze die lokale Grenze ist. Die Grenze, die verwendet wird, ist kleiner als die lokale CCB-Grenze und die CCB-Grenze des PEP-Endpunkt-Peers.

[0172] Obwohl im Allgemeinen ein TCP- oder PBP-Sender eigentlich mit einem TCP- oder PBP-Empfänger umgehen kann, der sein bekannt gegebenes Fenster verkleinert (d.h. ein neues Fenster sendet, das kleiner ist als das vorherige Fenster, minus etwaiger Daten, die in dem Fenster verschickt werden), arbeitet das Protokoll ineffizient, wenn dies der Fall ist. Daher werden TCP- und PBP-Empfänger durch ihre Protokolldefinitionen so eingeschränkt, dass sie ein zuvor bekannt gegebenes Fenster nicht verkleinern dürfen. Angesichts dieses Falles sollte ein TCP- oder PBP-Empfänger im Allgemeinen sein Empfangsfenster nicht auf die Gesamtmenge an verfügbarem Pufferplatz setzen, da andere Anwender dieses Pufferplatzes bewirken können, dass die Menge an Pufferplatz so weit schrumpft, dass sie nicht mehr der Steuerung der jeweiligen TCP- oder PBP-Verbindung unterliegt. Anders ausgedrückt, das Verschicken eines großen Fensters reduziert die Flexibilität in Bezug auf die Fähigkeit, auf eine reduzierte Pufferverfügbarkeit durch Verlangsamung des TCP- oder PBP-Senders zu reagieren. Daher ist es wünschenswert, dass maximale bekannt gegebene TCP- und PBP-Fenstergrößengrenzen erzwungen werden. Diese Grenzen stellen das größte Fenster dar, das ein TCP- oder PBP-Empfänger dem Sender in jedem Segment, das zu dem Sender geschickt wird, bekannt gibt. Es sei jedoch darauf hingewiesen, dass, falls der verfügbare Pufferplatz klein ist, kleinere Fenster (einschließlich 0) verschickt werden können. Dagegen ist es wichtig, dass, wenn ausreichend Pufferplatz zur Verfügung steht, das Fenster, das von einem TCP- oder PBP-Empfänger bekannt gegeben wird, groß genug ist, um die Bandbreite * Verzögerungsprodukt (d.h. die Größe der Leitung dividiert durch die Umlaufzeit der Leitung), welche auf die Verbindung zutrifft, abzudecken (damit der TCP- oder PBP-Sender die Verbindungsleitung voll halten). Da die Umlaufzeit von Netzwerk zu Netzwerk unterschiedlich sein könnte, ist die Verwendung von fest codierten Werten für die Fenstergrößen-Obergrenzen unerwünscht. Daher können diese Grenzen als Teil eines PEP-Endpunktprofils eines PEP-Endpunkts konfiguriert werden.

[0173] Das PEP-Endpunktprofil kann eine TCP-Fenstergrößen-Obergrenze und eine PBP-Fenstergrößen-Obergrenze einschließen. Da die meisten TCP-Verbindungen lokal für den PEP-Endpunkt **210** sind (verbunden über Ethernet), kann eine niedrige TCP-Fenstergrößen-Obergrenze die Umlaufzeit für die meisten Fälle abdecken. Daher könnte in diesem Fall die maximale TCP-Fenstergrößen-Voreinstellung auf 8 KB gesetzt werden. Aufgrund der Vielfalt von möglichen Verknüpfungsgeschwindigkeiten, die für PBP-Verbindungen möglich sind, ist ein Voreinstellungswert, der für die meisten Situationen passt, nicht möglich.

[0174] Die folgende Erörterung beschreibt die Berechnungen, die von der Plattformumgebung **210**, dem TCP-Spoofing-Kernel **280** und dem Backbone-Protocol-Kernel **282** durchgeführt werden, um verfügbaren Pufferplatz in bekannt gegebene Empfangsfenstergrößen umzuwandeln. Für jede Backbone-Verbindung leitet die Plattformumgebung **210**, wie oben gezeigt, die Menge an Pufferplatz, die in WAN-zu-LAN-Richtung für die Verbindung genutzt werden kann, durch Multiplizieren des WAN-zu-LAN-Pufferplatzwerts pro Peer mit dem Prozentanteil der Ressourcen pro Peer, die dieser Backbone-Verbindung zugeteilt wurden, ab. Der resultierende Wert wird dann als die Obergrenze für den WAN-zu-LAN-Pufferplatz für diese Backbone-Verbindung verwendet. Da die WAN-zu-LAN-Pufferplatzwerte pro Peer sich für jeden Peer unterscheiden können, kann die Plattformumgebung **210** die entsprechende Grenze für die Menge an LAN-zu-WAN-Pufferplatz nicht direkt berechnen, obwohl die PEP-Endpunkt-Peers sich den gleichen Prozentanteil an Ressourcenparametern teilen können;

stattdessen wird der Wert vom TCP-Spoofing-Kernel **280** bereitgestellt. Die Umgebung **210** stellt dem TSK **280** die WAN-zu-LAN-Puffergrenze (und die Grenze für die lokale Zahl der CCBs) bereit, wenn die Backbone-Verbindung geöffnet wird. Der TSK **280** schickt dann die Grenze an seinen TSK-Peer in einer TSK-Peer-Parameternachricht. Wenn der TSK **280** eine TPP-Nachricht erhält, extrahiert er die WAN-zu-LAN-Pufferplatzgrenze des Peers aus der Nachricht und gibt sie an seine Umgebung weiter. Die Umgebung nutzt die WAN-zu-LAN-Pufferplatzgrenze des Peers als ihre lokale LAN-zu-WAN-Pufferplatzgrenze. Wenn eine Backbone-Verbindung das erste Mal geöffnet wird, werden während des Wartens auf den Empfang einer TPP-Nachricht vom Peer die LAN-zu-WAN-Pufferplatzgrenze und die Grenze der Zahl der CCBs bei 0 initialisiert. Dies verhindert, dass der Bandbreitenbedarf von TCP-Verbindungen reduziert wird, bis gültige Peer-Parameterinformationen empfangen werden. Wie bereits beschrieben, zählt die Plattformumgebung die Zahl der LAN-zu-WAN-Puffer **1201** und der WAN-zu-LAN-Puffer **1203**, die sie jeder Backbone-Verbindung zugewiesen hat, im ECB der Backbone-Verbindung. Wenn ein Puffer zugewiesen wird, wird die angemessene Verwendungszählung inkrementiert. Wenn ein Puffer freigegeben wird, wird der Backbone-Verbindungs-Handle, der von der Umgebung in dem Puffer gespeichert wurde, verwendet, um die richtige Verwendungszählung für die Dekrementierung zu finden. Wenn vom TSK **280** oder vom BPK **282** aufgefordert, schickt die Umgebung **210** den gerade verfügbaren LAN-zu-WAN- oder WAN-zu-LAN-Pufferplatz für eine Backbone-Verbindung zurück. In einer Plattform (z.B. der PES Remote), wo kleine, verkettete Puffer verwendet werden, muss die Plattformumgebung **210** ihre Pufferzählung mit der Zahl der Puffer, die benötigt sind, um ein IP-Paket maximaler Größe aufzunehmen, normalisieren. TSK **280** und BPK **282** verwenden diese Werte, um Fenstergrößen wie folgt zu berechnen:

$$A_{pi}^{W2L} = B_{pi}^{W2L} - U_{pi}^{W2L}$$

$$A_{pi}^{L2W} = B_{pi}^{L2W} - U_{pi}^{L2W},$$

wobei B_{pi}^{W2L} die berechnete WAN-zu-LAN-Pufferplatzgrenze für die Backbone-Verbindung zum Peer „p" bei Priorität „i" ist, B_{pi}^{L2W} die erlernte LAN-zu-WAN-Pufferplatzgrenze für die Backbone-Verbindung zum Peer „p" bei Priorität „i" ist, U_{pi}^{W2L} der WAN-zu-LAN-Pufferplatz ist, der für die Backbone-Verbindung zum Peer „p" bei Priorität „i" verwendet wird, U_{pi}^{L2W} der LAN-zu-WAN-Pufferplatz ist, der für die Backbone-Verbindung zum Peer „p" bei Priorität „i" verwendet wird, A_{pi}^{W2L} der WAN-zu-LAN-Pufferplatz ist, der für die Backbone-Verbindung zum Peer „p" bei Priorität „i" zur Verfügung steht, und A_{pi}^{W2L} der LAN-zu-WAN-Pufferplatz ist, der für die Backbone-Verbindung zum Peer „p" bei Priorität „i" zur Verfügung steht.

[0175] Zusätzlich zur Menge des verfügbaren Pufferplatzes kann es für einen PEP-Endpunkt **210** wünschenswert sein, andere Faktoren in Betracht zu ziehen, wenn die bekannt zu gebenden Fenstergrößen bestimmt werden. Insbesondere kann die aktuelle Latenz (im Allgemeinen anhand der Länge der Warteschlange) für die WAN-Schnittstelle **1215** ein wichtiger Faktor sein, da diese Schnittstelle **1215** ein Multiplexing-Punkt für den Verkehr vieler konkurrierender Flüsse ist, insbesondere im Hub. Tatsächlich schließt die PEP-TCP-Spoofing-Implementierung die Fähigkeit zur Überwachung der Warteschlangenlatenz und zur Anpassung von TCP-Fenstergrößenbekanntgaben ein, wenn die Warteschlangenlatenz zunimmt und abnimmt. Wenn vom Operator ermöglicht, kann die Umgebung **210** die Warteschlangenlatenz verfolgen und diesen Wert verwenden, um einen aktuellen Flusssteuerungsfaktor zu bestimmen. In einem Ausführungsbeispiel kann der Flusssteuerungsfaktor als Prozentanteil von 0 % bis 100 % verfolgt werden. Wenn die Latenz um einen bestimmten, vom Operator definierten Wert zunimmt, kann die Umgebung **210** den Flusssteuerungsfaktor verkleinern. Wenn die Latenz um einen bestimmten, vom Operator definierten Wert abnimmt, kann die Umgebung **210** den Flusssteuerungsfaktor vergrößern. Nominal können Inkremente von 5 % verwendet werden, um den Flusssteuerungsfaktor nach unten und nach oben anzupassen. Die exakten Inkrementierungseinheiten sind jedoch nicht besonders wichtig. Sobald die Plattformumgebung **210** eine Anforderung für die Menge an verfügbarem Pufferplatz in LAN-zu-WAN-Richtung empfängt, multipliziert sie das Ergebnis (bestimmt wie oben) mit dem Flusssteuerungsfaktor, wie nachstehend dargestellt.

$$A_{pi}^{L2W} = F \cdot A_{pi}^{L2W},$$

wobei F der aktuelle Flusssteuerungsfaktor, ausgedrückt als Prozentanteil, ist. Dies führt zu einem verringerten Input vom TCP-Host, der zum PEP-Endpunkt lokal ist, wenn die Latenz zunimmt.

[0176] Die Verwendung der Latenz, um Fenstergrößen anzupassen, kann auf PBP-Fenster angewendet werden. Insbesondere können Warteschlangenlatenzen, die mit der Verschickung von Verkehr in WAN-zu-LAN-Richtung in Beziehung stehen, verwendet werden, um die Fenster anzupassen, die vom PBP bekannt gegeben werden.

[0177] Fig. 29 zeigt ein Gleitfenster, das vom PBP verwendet wird, gemäß einer Ausführungsform der vorliegenden Erfindung. Wie der TCP verwendet der PBP ein gleitendes Fenster **2901**, um den aktuellen akzeptablen Bereich von Sequenznummern zu be-

stimmen. Wie dargestellt, ist der linke Rand im Sender die letzte bestätigte Sequenznummer plus eins (Snd_Una). Der rechte Rand ist gleich dem linken Rand plus die vom Empfänger bekannt gegebenen Fenstergröße. Der Sender kann das Fenster **2901** füllen und nach Füllen des Fensters **2901** muss er auf eine Bestätigung warten, um neue Pakete übertragen zu können. Falls das Fenster voll ist und der Sender neue Daten zum Versenden erhält, muss er diese Daten für die spätere Übermittlung nach Gleiten des Fensters **1901** in die Warteschlange stellen. Der Empfänger betrachtet das Fenster mittels Rcv_Nxt für den linken Rand anstelle von Snd_Una. Falls ein empfangenes Paket eine Sequenznummer innerhalb des Fensters aufweist, wird es bestätigt. Falls es dem linken Rand des Fensters **2901** entspricht, wird ein kumulativer ACK verwendet, der das Fenster **2901** um 1 nach unten schiebt.

[0178] Wenn der TCP-Spoofing-Kernel (TSK) **280** eine Fenstergröße bestimmen muss, um ein TCP-Segment bekannt zu geben, beginnt der TSK **280** durch Aufrufen der Plattformumgebung **210**, um die Verfügbarkeit des aktuellen LAN-zu-WAN-Pufferplatzes für die Backbone-Verbindung zu erhalten, die mit der bandbreitenbedarfsreduzierten TCP-Verbindung assoziiert ist. Der TSK **280** teilt dann diese Zahl durch die Zahl der TCP-Verbindungen, die derzeit die Backbone-Verbindung nutzen. Der TSK **280** verfolgt die Zahl der TCP-Verbindungen, die eine Backbone-Verbindung nutzen, im TCB der Backbone-Verbindung, wobei er die Zählung inkrementiert, sobald ein CCB zugeordnet wird, und die Zählung dekrementiert, sobald ein CCB freigegeben wird. Der TSK **280** wandelt diesen Wert dann durch Multiplizieren der Zahl der Puffer mit den vom lokalen Host verwendeten MSS von Puffern in Byte um, um TCP-Segmenten an den TSK **280** zu schicken. Dieser Wert stellt die mögliche Fenstergröße dar, die bekannt gegeben werden kann. Der TSK **280** muss jedoch zwei zusätzliche Überprüfungen vornehmen, bevor er diesen Wert verwendet. Zuerst wird der mögliche Wert mit der Fenstergrößengrenze verglichen. Falls der mögliche Wert größer ist als die Fenstergrößengrenze, wird stattdessen die Fenstergrößengrenze bekannt gegeben. Falls der mögliche Wert kleiner ist als die Fenstergrößengrenze, führt der TSK **280** dann eine Überprüfung durch, um zu bestimmen, ob eine Bekanntgabe des möglichen Werts das Fenster auf einen Wert schrumpfen lassen würde, der kleiner als der zuvor bekannt gegebene ist (d.h. der den rechten Rand des rotierenden Fensters nach links verschieben würde). Wie bereits angegeben, sollte ein TCP-Empfänger sein Fenster **2901** nicht verkleinern; falls der potentielle Fensterwert das Fenster **2901** schrumpfen lassen würde, gibt der TSK **280** daher stattdessen das kleinste mögliche Fenster **2901** bekannt, welches das zuvor bekannt gegebene Fenster nicht schrumpfen lässt (d.h. den Wert, der dafür steht, den rechten Rand des Fensters **2901** an Ort und Stelle zu lassen). Die Berechnung des bekannt gegebenen TCP-Fensters **2901** ist wie folgt:

$$W_{TC} = A_{pi}^{L2W}/K_{pi} \cdot MSS$$

$$W_{TA} = MAX(MIN(W_{TC}, W_{TL}), W_{TR}),$$

wobei K_{pi} die aktuelle Zahl von TCP-Verbindung ist, die die Backbone-Verbindung zum Peer „p" bei Priorität „i" nutzt, W_{TC} das berechnete TCP-Fenster **2901** ist, W_{TR} das TCP-Fenster ist, das von dem Platz dargestellt wird, der vom zuvor bekannt gegebenen Fenster bleibt (d.h. aufgrund des letzten bekannt gegebenen „rechten Rands"), W_{TL} die konfigurierte bekannt gegebene TCP-Fenster-Obergrenze ist, W_{TA} das TCP-Fenster ist, das tatsächlich bekannt gegeben wird, und MSS die TCP-Verbindungs-MSS ist.

[0179] Die PBP-Fensterberechnungen gleichen den TCP-Fensterberechnungen, abgesehen davon, dass keine Notwendigkeit zum Umwandeln des Fensters in Byte bestehen muss. Wenn der Backbone-Protokoll-Kernel **282** eine Fenstergröße zur Bekanntgabe in einem PBP-Segment bestimmen muss, beginnt der BPK mit dem Aufruf der Plattformumgebung **210**, um die aktuelle Verfügbarkeit des WAN-zu-LAN-Pufferplatzes für die Backbone-Verbindung zu erhalten. Dieser Wert stellt die mögliche Fenstergröße dar, die bekannt gegeben werden kann. Jedoch muss der BPK zwei zusätzliche Überprüfungen durchführen, bevor dieser Wert verwendet wird. Zuerst wird der mögliche Wert mit der Fenstergrößengrenze verglichen. Falls der mögliche Wert größer ist als die Fenstergrößengrenze, wird stattdessen die Fenstergrößengrenze bekannt gegeben. Falls der mögliche Wert kleiner ist als die Fenstergrößengrenze, fuhrt der BPK **282** dann eine Überprüfung durch, um zu bestimmen, ob der mögliche Wert das Fenster auf einen Wert schrumpfen lässt, der kleiner ist als der zuvor bekannt gegebene (d.h. ob er den rechten Rand des sich drehenden Fensters nach links bewegen würde). Wie oben beschrieben, sollte ein PBP-Empfänger sein Fenster **2901** nicht verkleinern. Wenn der potentielle Fensterwert das Fenster **2901** schrumpfen lassen würde, gibt der BPK **282** stattdessen das kleinste mögliche Fenster **2901** bekannt, das das zuvor bekannt gegebene Fenster **2901** nicht schrumpfen lässt (d.h. den Wert, der die Beibehaltung des rechten Rands des Fensters an Ort und Stelle darstellt). Die Berechnung des bekannt gegebenen PBP-Fensters ist wie folgt:

$$W_{PC} = A_{pi}^{W2L}$$

$$W_{PA} = MAX(MIN(W_{PC}, W_{PL}), W_{PR})$$

wobei W_{PC} das berechnete PBP-Fenster **2901** ist, W_{PR} das PBP-Fenster ist, das durch den Platz dargestellt wird, der vom zuvor bekannt gegebenen Fenster bleibt (d.h. aufgrund des letzten bekannt gegebe-

nen „rechten Rands"), W_{PL} die konfigurierte bekannt gegebene PBP-Fenster-Obergrenze ist und W_{PA} das PBP-Fenster ist, das tatsächlich bekannt gegeben wird.

[0180] Fig. 30 stellt ein Computersystem **3001** dar, mit dem eine Ausführungsform gemäß der vorliegenden Erfindung implementiert werden kann. Ein solches Computersystem **3001** kann als Server konfiguriert sein, um einen Code auszuführen, der die PEP-Funktionen des PEP-Endpunkts **210** durchführt wie vorstehend erörtert. Das Computersystem **3001** schließt einen Bus **3003** oder einen anderen Kommunikationsmechanismus zur Übermittlung von Informationen und einen Prozessor **3005** ein, der mit dem Bus **3003** verkoppelt ist, um diese Informationen zu verarbeiten. Das Computersystem **3001** schließt auch einen Hauptspeicher **3007**, wie einen Schreib/Lese-Speicher (RAM) oder eine andere dynamische Speichereinrichtung ein, der mit dem Bus **3003** gekoppelt ist, um Informationen und Befehle zu speichern, die vom Prozessor **3005** ausgeführt werden sollen. Darüber hinaus kann der Hauptspeicher **3007** zum Speichern vorübergehender Variablen oder anderer Zwischeninformationen während der Ausführung der Befehle, die vom Prozessor **3005** ausgeführt werden sollen, verwendet werden. Insbesondere können PEP-Steuerblöcke im Hauptspeicher **3007** gespeichert werden. Das Computersystem **3001** schließt ferner einen Festwertspeicher (ROM) **3009** oder eine andere statische Speichereinrichtung ein, die mit dem Bus **3003** gekoppelt ist, um statische Informationen und Befehle für den Prozessor **3005** zu speichern. Eine Speichereinrichtung **3011**, wie eine Magnetplatte oder eine optische Platte, wird bereitgestellt und mit dem Bus **3003** gekoppelt, um Informationen und Instruktionen zu speichern.

[0181] Das Computersystem **3001** kann über einen Bus **3003** mit einer Anzeige **3013**, wie einer Kathodenstrahlröhre (CRT) zum Anzeigen von Informationen für den einen Computernutzer gekoppelt werden. Eine Eingabeeinrichtung **3015**, die alphanumerische oder andere Tasten aufweist, ist mit dem Bus **3003** verkoppelt, um Informationen und Befehlsauswahlen an den Prozessor **3005** zu übermitteln. Eine andere Art von Anwender-Eingabeeinrichtung ist eine Cursor-Steuerung **3017**, wie eine Maus, ein Trackball oder Cursor-Lenkungstasten zum Übermitteln von Lenkungsinformationen und Befehlsauswahlen an den Prozessor **3005** und zum Steuern der Cursorbewegung auf der Anzeigen **3013**.

[0182] Ausführungsformen sind auf die Verwendung eines Computersystems **3001** bezogen, um die PEP-Funktionen des PEP-Endpunkts **210** auszuführen. Gemäß einer Ausführungsform wird dieser automatische Aktualisierungsansatz durch ein Computersystem **3001** als Antwort darauf bereitgestellt, dass der Prozessor **3005** eine oder mehrere Sequenzen eines oder mehrerer Befehle, die im Hauptspeicher **3007** enthalten sind, ausführt. Diese Befehle können von einem anderen computerlesbaren Medium, wie einer Speichereinrichtung **3011** in den Hauptspeicher eingelesen werden. Die Ausführung der Befehlssequenzen, die im Hauptspeicher **3007** enthalten sind, bewirkt, dass der Prozessor **3005** die hierin beschriebenen Prozessorschritte ausführt. Ein oder mehrere Prozessoren in einer Multiprozessor-Umgebung können ebenfalls verwendet werden, um die Befehlssequenzen auszuführen, die im Hauptspeicher **3007** enthalten sind. In alternativen Ausführungsformen können fest verdrahtete Schaltungen anstelle von oder in Kombination mit Software-Befehlen verwendet werden. Somit sind die Ausführungsformen nicht auf eine bestimmte Kombination von Hardware-Verdrahtung und Software beschränkt.

[0183] Der Ausdruck „computerlesbares Medium", wie hierin verwendet, bezieht sich auf jedes Medium, das an der Bereitstellung von Befehlen an den Prozessor **3005** für die Ausführung der PEP-Funktionen des PEP-Endpunkts **210** beteiligt sind. Diese Medien können viele Formen haben, einschließlich von nicht-flüchtigen Medien, flüchtigen Medien und Übertragungsmedien, aber nicht darauf beschränkt. Nicht-flüchtige Medien schließen beispielsweise optische oder magnetische Scheiben, wie eine Speichervorrichtung **3011** ein. Flüchtige Medien schließen dynamische Speicher, wie einen Hauptspeicher **3007** ein. Übertragungsmedien schließen Koaxialkabel, Kupferdraht und Faseroptik, einschließlich der Drähte, aus denen der Bus **3003** besteht, ein. Übertragungsmedien können auch die Form von akustischen oder von Lichtwellen haben, so wie diejenigen, die während Radiowellen- und Infrarotdatenübertragungen erzeugt werden.

[0184] Übliche Formen solcher computerlesbarer Medien schließen beispielsweise eine Floppy Disk, eine flexible Disk, eine Hard Disk, ein Magnetband oder ein anderes magnetisches Medium, eine CD-ROM, ein anderes optisches Medium, Lochkarten, Papierband, jedes andere physische Medium mit Lochmustern, einen RAM, einen PROM und einen EPROM, einen FLASH-EPROM, jeden anderen Speicherchip oder jede andere Speicherkassette, eine Trägerwelle, wie nachstehend beschrieben, oder jedes andere Medium, das ein Computer lesen kann, ein.

[0185] Es können verschiedene Formen von computerlesbaren Medien an der Beförderung einer oder mehrerer Sequenzen eines oder mehrerer auszuführender Befehle an den Prozessor **3005** beteiligt sein. Beispielsweise können sich die Befehle auf einer Magnetplatte oder einem entfernten Computer befinden. Der entfernte Computer kann die Befehle, die sich auf die Ausführung der PEP-Funktionen des

PEP-Endpunkts **210** beziehen, in seinen dynamischen Speicher laden und die Befehle über eine Telefonleitung unter Verwendung eines Modems verschicken. Ein Modem, das sich lokal an einem Computersystem **3001** befindet, kann die Daten über die Telefonleitung empfangen und einen Infrarotüberträger verwenden, um die Daten in ein Infrarotsignal umzuwandeln. Ein Infrarotdetektor, der mit dem Bus **3003** gekoppelt ist, kann die Daten, die vom Infrarotsignal übermittelt werden, empfangen und die Daten auf den Bus **3003** übertragen. Der Bus **3003** übermittelt die Daten an den Hauptspeicher **3007**, von dem der Prozessor **3005** die Befehle ermittelt und ausführt. Die Befehle, die vom Hauptspeicher **3007** empfangen werden, können optional in der Speichereinrichtung **3011** gespeichert werden, entweder vor oder nach Ausführung durch den Prozessor **3005**.

[0186] Das Computersystem **3001** schließt auch eine oder mehrere Kommunikationsschnittstellen **3019** ein, die mit dem Bus **3003** gekoppelt sind. Die Kommunikationsschnittstellen **3019** stellen eine Zweiwege-Datenkommunikationskopplung an Netzwerkverknüpfungen **3021** und **3022** bereit, die mit einem Local Area Network (LAN) **3023** bzw. einem Wide Area Network (WAN) **3024** verbunden sind. Das WAN **3024** kann gemäß einer Ausführungsform der vorliegenden Erfindung, ein Satellitennetz sein. Die Kommunikationsschnittstelle **3019** kann eine Netzschnittstellenkarte sein, die an einem beliebigen paketvermittelten LAN angebracht wird. Als weiteres Beispiel kann eine Kommunikationsschnittstelle **3019** eine asymmetrische digitale Teilnehmerleitungs- (ADSL-) Karte, eine dienstintegrierende digitale Netzwerk- (ISDN-) Karte, ein Kabelmodem oder ein Modem, das für Datenübertragungsverbindungen mit einer entsprechenden Art von Telefonleitung sorgt, sein. Es können auch drahtlose Verknüpfungen implementiert werden. In jeder dieser Implementierungen verschickt und empfängt die Kommunikationsschnittstelle **3019** elektrische, elektromagnetische oder optische Signale, die digitale Datenströme übermitteln, welche verschiedene Arten von Informationen darstellen.

[0187] Die Netzwerkverknüpfung **3021** stellt in der Regel Datenübermittlungen über eines oder mehrere Netzwerke an andere Datenvorrichtungen bereit. Beispielsweise kann die Netzwerkverknüpfung **3021** eine Verbindung durch ein lokales Netze **3023** zu einem Host-Computer **3025** oder einer Datenausrüstung, die von einem Internet-Dienstanbieter (ISP) **3027** betrieben wird, bereitstellen. Der ISP **3027** seinerseits stellt Datenübermittlungsdienste über das Internet **505** bereit. Darüber hinaus ist das LAN **3023** mit einem Intranet **3029** verknüpft. Das Intranet **3029**, das LAN **3023** und das Internet **505** nutzen sämtlich elektrische, elektromagnetische oder optische Signale, welche die digitalen Datenströme übermitteln. Die Signale durch die verschiedenen Netzwerke und die Signale auf der Netzwerkverknüpfung **3021** und durch die Kommunikationsschnittstelle **3019**, welche die digitalen Daten zum und vom Computersystem **3001** übermitteln, sind Beispiele für Formen von Trägerwellen, die die Informationen transportieren.

[0188] Das Computersystem **3001** kann über das bzw. die Netzwerk(e), die Internet-Verknüpfung **3021** und die Kommunikationsschnittstelle **3019** Nachrichten versenden und Daten, einschließlich von Programm-Code, versenden und empfangen. In dem Internetbeispiel könnte ein Server **3031** einen angeforderten Code für ein Anwendungsprogramm durch das Internet **505**, den ISP **3027**, das LAN **3023** und die Kommunikationsschnittstelle **3019** übermitteln. Der empfangene Code kann vom Prozessor **3005** wie empfangen ausgeführt werden und/oder in einer Speichervorrichtung **3011** oder einem anderen nicht-flüchtigen Speicher für die spätere Ausführung gespeichert werden. Auf diese Weise kann das Computersystem **3001** einen Anwendungs-Code in Form einer Trägerwelle erhalten. Das Computersystem **3001** kann über das bzw. die Netzwerk(e), die Netzwerkverknüpfung **3021** und die Kommunikationsschnittstelle **3019** Benachrichtigungen übermitteln und Daten, einschließlich von Programm-Code, empfangen.

[0189] Die hierin beschriebenen Verfahren bieten mehrere Vorteile gegenüber Versuchen des Standes der Technik, die Netzwerkleistung zu verbessern, insbesondere in einem paketvermittelten Netzwerk wie dem Internet. Ein lokaler PEP-Endpunkt und ein entfernter PEP-Endpunkt kommunizieren, um den Austausch von Daten durch eine Backbone-Verbindung durch die Nutzung von leistungsverbessernden Funktionen zu optimieren. Dieser Ansatz minimiert auf vorteilhafte Weise die Netzwerklatenz.

[0190] Natürlich können zahlreiche Modifikationen und Variationen der vorliegenden Erfindung angesichts der obigen Lehren durchgeführt werden. Es ist daher selbstverständlich, dass innerhalb des Bereichs der beigefügten Ansprüche die Erfindung auch auf andere Weise als hierin speziell beschrieben ausgeführt werden kann.

Patentansprüche

1. Netzwerkeinrichtung zur Erzeugung von Betriebsverbesserungen eines Kommunikationsnetzwerkes (**100**), wobei die Netzwerkeinrichtung folgendes enthält:
Mittel zum Empfang von Nachrichten gemäß einem vorgeschriebenen Protokoll;
Mittel zur Anpassung an unterschiedliche Nachrichtentypen mit veränderlichen Kopffeldlängen durch Verbrauch des Raumes eines ausdehnbaren Kopffeldbereiches (**1607**);
Mittel zum Ausdehnen der Größe des ausdehnbaren

Kopffeldbereiches (**1607**), wenn das Kopffeld der empfangenen Nachrichten nicht durch die vorliegende Größe des ausdehnbaren Kopffeldbereiches (**1607**) aufgenommen werden kann;
Mittel zur Verarbeitung der Nachrichten zur Erzeugung der Betriebsverbesserungsfunktion;
eine Mehrzahl von Puffern (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**), welche so konfiguriert sind, daß sie die empfangenen Nachrichten und die Nachrichten speichern, welche durch die Verarbeitungsmitttel erzeugt worden sind, wobei ein Teil der Mehrzahl der Puffer (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**) durch die Verarbeitungsmittel basierend auf der Durchführung einer bestimmten der die Betriebsverbesserung bewirkenden Funktionen anteilig benutzt werden.

2. Netzwerkeinrichtung nach Anspruch 1, bei welcher die Mittel zum Empfang der Nachrichten eine Mehrzahl von Kommunikationsschnittstellen (**3019**) enthalten, welche so konfiguriert sind, daß sie Nachrichten gemäß dem vorbestimmten Protokoll empfangen und weitergeben.

3. Netzwerkeinrichtung nach Anspruch 2, bei welcher eine Anzahl von Modulen (**280**, **282**, **284**, **286**, **290**, **292**) ein Vortäuschungsmodul oder Spoofingmodul (**280**), welches zur Durchführung einer selektiven Vortäuschung konfiguriert ist, ein Verbindungsmodul, das zur Multiplexverteilung einer Mehrzahl von Verbindungen über eine gemeinsame Rückgratverbindung, ein Prioritäts-Zuteilungsmodul zur Prioritätszuteilung des Zuganges zu der Rückgratverbindung und ein Übertragungswegauswahlmodul enthält, das zur Bestimmung eines Übertragungsweges unter einer Mehrzahl von Übertragungswegen zur Übertragung der empfangenen Nachrichten ausgebildet ist.

4. Netzwerkeinrichtung nach Anspruch 3, bei welcher die Kommunikationsschnittstellen eine Lokalbereichsnetzwerk-Schnittstelle und eine Weitbereichsschnittstelle enthalten, wobei einer der Mehrzahl von Puffern (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**) als Puffer zwischen dem Lokalbereichsnetzwerk und dem Weitbereichsnetzwerk ausgebildet ist, welcher die empfangenen Nachrichten in der Richtung vom Lokalbereichsnetzwerk zum Weitbereichsnetzwerk speichert, während ein anderer der Mehrzahl von Puffern (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**) als ein Puffer vom Weitbereichsnetzwerk zum Nahbereichsnetzwerk ausgebildet ist, der die empfangenen Nachrichten mit Bezug auf eine Richtung von dem Weitbereichsnetzwerk zu dem dem Lokalbereichsnetzwerk speichert.

5. Netzwerkeinrichtung nach Anspruch 4, wobei das Weitbereichsnetzwerk ein Satellitennetzwerk (**100**) ist.

6. Netzwerkeinrichtung nach Anspruch 2, bei welcher jede der erzeugten Nachrichten folgendes enthält:
ein spezifisches Kopffeld (**1401**), welches plattformspezifische Information speichert;
ein allgemeines Kopffeld (**1403**), welches Informationen speichert, die der Mehrzahl von Modulen (**280**, **282**, **284**, **286**, **290**, **292**) bekannt sind;
ein Nutzdatenfeld (**1503**); und
ein Versatzfeld (**1601b**), welches den Start des Nutzdatenfeldes (**1503**) anzeigt.

7. Netzwerkeinrichtung nach Anspruch 6, bei welchem das allgemeine Kopffeld (**1403**) folgendes enthält:
ein Flaggenfeld oder Markenfeld (**1601**), welches die Richtung des Nachrichtenflusses anzeigt;
ein Verbindungshandhabungsfeld (**1603**), das die Handhabung einer Rückgratverbindung bezeichnet; und
ein eigentümerspezifisches Feld (**1605**), das eine eigentümerspezifische Kopffeldinformation speichert.

8. Netzwerkeinrichtung nach Anspruch 2, bei welcher das bestimmte Protokoll das Transmission Controll Protocoll (TCP) ist.

9. Verfahren zur Erzeugung von Betriebsverbesserungen eines Kommunikationsnetzwerkes (**100**), wobei das Verfahren folgendes umfasst:
Empfangen von Nachrichten gemäß einem vorgeschriebenen Protokoll;
Anpassung an unterschiedliche Nachrichtentypen mit veränderlichen Kopffeldlängen durch Verbrauch des Raumes eines ausdehnbaren Kopffeldbereiches (**1607**) entsprechend einem aus einer Anzahl von Puffern (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**);
Ausdehnen der Größe des ausdehnbaren Kopffeldbereiches (**1607**), wenn die Kopffelder der empfangenen Nachrichten nicht durch die augenblickliche Größe des ausdehnbaren Kopffeldbereiches (**1607**) aufgenommen werden können;
Verarbeiten der Nachrichten zur Erzielung der Betriebsverbesserungsfunktionen über eine Anzahl von Modulen (**280**, **282**, **284**, **286**, **290**, **292**); und
Speichern der empfangenen Nachrichten und der Nachrichten, welche durch einen der Mehrzahl von Modulen (**280**, **282**, **284**, **286**, **290**, **292**) erzeugt worden sind, in einer Anzahl von Puffern (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**), wobei der Anteil der Anzahl von Puffern (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**) durch die Anzahl von Modulen (**280**, **282**, **284**, **286**, **290**, **292**) basierend auf der Durchführung einer bestimmten der die Betriebsverbesserung bewirkenden Funktionen anteilmäßig benutzt wird.

10. Verfahren nach Anspruch 9, welches weiter folgendes umfasst:
Durchführen einer selektiven Spoofing-Funktion;
Multiplexverarbeitung einer Mehrzahl von Verbindungen über eine gemeinsame Rückgratverbindung;

Prioritätsverteilung des Zuganges zu der Rückgratverbindung; und
Bestimmung eines Übertragungsweges aus einer Mehrzahl von Übertragungswegen zur Übertragung der empfangenen Nachrichten.

11. Verfahren nach Anspruch 9, bei welchem der Schritt des Empfangens durch eine Kommunikationsschnittstelle durchgeführt wird, welche eine Lokalbereichsnetzwerkschnittstelle und/oder eine Weitbereichsnetzwerkschnittstelle, einen der Mehrzahl von Puffern (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**), welche als Puffer vom Lokalbereichsnetzwerk zum Weitbereichsnetzwerk ausgebildet sind, wobei der Puffer die empfangenen Nachrichten in der Richtung vom Lokalbereichsnetzwerk zum Weitbereichsnetzwerk speichert, einen anderen der Mehrzahl von Puffern (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**), welcher als Puffer vom Weitbereichsnetzwerk zum Lokalbereichsnetzwerk ausgebildet ist, der die empfangenen Nachrichten mit Bezug auf eine Richtung vom Weitbereichsnetzwerk zum Nahbereichsnetzwerk ausgebildet ist, enthält.

12. Verfahren nach Anspruch 11, wobei das Weitbereichsnetzwerk ein Satellitennetzwerk (**100**) ist.

13. Verfahren nach Anspruch 9, bei welchem jede der erzeugten Nachrichten folgendes umfasst:
ein spezifisches Kopffeld (**1401**), welches plattformspezifische Information speichert;
ein allgemeines Kopffeld (**1403**), welches Informationen speichert, die für die Mehrzahl von Modulen (**280**, **282**, **284**, **286**, **290**, **292**) bekannt sind;
ein Nutzdatenfeld (**1503**); und
ein Versatzfeld (**1601b**), welches den Beginn des Nutzdatenfeldes (**1503**) anzeigt.

14. Verfahren nach Anspruch 13, bei welchem das allgemeine Kopffeld (**1403**) folgendes enthält:
ein Flaggenfeld (**1601**), das die Richtung des Nachrichtenflusses anzeigt;
ein Verbindungshandhabungsfeld (**1603**), das die Handhabung einer Rückgratverbindung beschreibt; und
Ein eigentümerspezifisches Feld (**1605**), das eigentümerspezifische Kopfdaten speichert.

15. Verfahren nach Anspruch 9, bei welchem das vorbestimmte Protokoll in dem Schritt des Empfangens das Transmission Controll Protocoll (TCP) ist.

16. Computerlesbares Medium, welches eine Folge oder mehrere Folgen von einem Befehl oder mehreren Befehlen zur Erzeugung von Betriebsverbesserungen eines Kommunikationsnetzwerkes (**100**) trägt, wobei die eine Folge oder die mehreren Folgen von einem oder mehreren Befehlen Befehle enthält bzw. enthalten, welche bei Durchführung durch einen oder mehrere Prozessoren diesen bzw. diese zu folgenden Schritten veranlassen:
Empfang von Nachrichten gemäß einem vorgeschriebenen Protokoll;
Anpassung an unterschiedliche Nachrichtentypen mit veränderlicher Kopffeldlänge durch Aufbrauch des Raumes eines ausdehnbaren Kopffeldbereiches (**1607**) entsprechend einem einer Mehrzahl von Puffern (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**);
Ausdehnen der Größe des ausdehnbaren Kopffeldbereiches (**1607**), wenn die Kopffelder der empfangenen Nachrichten nicht durch die gegenwärtige Größe des ausdehnbaren Kopffeldbereiches (**1607**) aufgenommen werden können;
Verarbeiten der Nachrichten zur Erzeugung der Verarbeitungsverbesserungsfunktionen über eine Mehrzahl von Modulen (**280**, **282**, **284**, **286**, **290**, **292**), wobei die Nachrichten ein Kopffeld veränderlicher Länge enthalten; und
Speichern der empfangenen Nachrichten und der Nachrichten, welche durch einen der Mehrzahl von Modulen (**280**, **282**, **284**, **286**, **290**, **292**) erzeugt werden, in einer Mehrzahl von Puffern (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**),
wobei ein Anteil der Mehrzahl von Puffern (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**) durch die Mehrzahl von Modulen (**280**, **282**, **284**, **286**, **290**, **292**) basierend auf der Durchführung einer bestimmten der die Betriebsweise verbessernden Funtktionen anteilsmäßig benutzt wird.

17. Computerlesbares Medium nach Anspruch 16, wobei der eine oder die mehreren Prozessoren weiter folgende Schritte durchführen:
Durchführen einer selektiven Spoofing-Funktion;
Multiplexverarbeitung einer Mehrzahl von Verbindungen über eine gemeinsame Rückgratverbindung;
Prioritätsverteilung des Zugriffes zu der Rückgratverbindung; und
Bestimmen eines Übertragungsweges aus einer Mehrzahl von Übertragungswegen zur Übertragung der empfangenen Nachrichten.

18. Computerlesbares Medium nach Anspruch 16, wobei der Schritt des empfangens durch eine Kommunikationsschnittstelle durchgeführt wird, welche eine Lokalbereichsschnittstelle (LAN) und/oder eine Weitbereichsschnittstelle (WAN), ferner einen aus einer Mehrzahl von Puffern (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**), der als ein LAN-zu-WAN-Puffer bezeichnet ist, welcher die empfangenen Nachrichten mit Bezug auf eine LAN-zu-WAN-Richtung speichert, einen anderen aus der Mehrzahl von Puffern Puffern (**1201**, **1203**, **1205**, **1207**, **1209**, **1211**), der als ein WAN-zu-LAN-Puffer bezeichnet ist, welcher die empfangenen Nachrichten in der WAN-zu-LAN-Richtung speichert, enthält.

19. Computerlesbares Medium nach Anspruch 18, wobei das weitbereichsnetzwerk (WAN) ein Satelitennetzwerk (**100**) ist.

20. Computerlesbares Medium nach Anspruch 16, bei welchem jede der erzeugten Nachrichten folgendes enthält:
Ein spezifisches Kopffeld (**1401**), das plattformspezifische Information speichert;
Ein allgemeines Kopffeld (**1403**), welches Information speichert, die für eine Mehzahl von Modulen (**280**, **282**, **284**, **286**, **290**, **292**) bekannt ist;
Ein Nutzdatenfeld (**1503**); und
ein Versatzfeld (**1601b**), welches den Beginn des Nutzdatenfeldes (**1503**) anzeigt.

21. Computerlesbares Medium nach Anspruch 20, bei welchem das allgemeine Kopffeld (**1403**) folgendes umfaßt:
ein Flaggenfeld oder Markenfeld (**1601**), welches die Richtung des Nachrichtenflusses anzeigt;
ein Verbindungshandhabungsfeld (**1603**), welches die Handhabung einer Rückgratverbindung beschreibt; und
ein eigentümerspezifisches Feld (**1605**), welches eine eigentümerspezifische Kopfinformation speichert.

22. Computerlesbares Medium nach Anspruch 16, bei welchem, das vorgeschriebene Protokoll in dem Empfangsschritt das Transmission Control Protocol (TCP) ist.

23. Ein Digitalsignal zur Vermittlung von Information zur Erzeugung einer Betriebsverbesserung in einem Kommunikationsnetzwerk (**100**), wobei das Signal eine Struktur aufweist, welche ein Kopfteil mit veränderlicher Länge besitzt und die Struktur folgendes enthält:
ein spezifisches Kopffeld (**1401**), welches eine plattformspezifische Information speichert;
ein allgemeines Kopffeld (**1403**), das Information speichert, welche für eine Mehrzahl von Modulen (**280**, **282**, **84**, **86**, **89**) bekannt ist;
ein Nutzdatenfeld (**1503**);
ein Versatzfeld (**1601b**), welches den Beginn des Nutzdatenfeldes (**1503**) anzeigt; und
einen ausdehnbaren Kopffeldbereich (**1607**), welcher sich der veränderlichen Länge der Kopfdateninformation anpaßt.

24. Digitalsignal nach Anspruch 23, bei welchem das allgemeine Kopffeld (**1403**) folgendes umfaßt:
ein Flaggenfeld oder Markenfeld (**1601**), das die Richtung des Nachrichtenflusses anzeigt;
ein Verbindungshandhabungsfeld (**1603**), welches die Handhabung einer Rückgratverbindung beschreibt; und
ein eigentümerspezifisches Feld (**1605**), welches eine eigentümerspezifische Kopfinformation speichert.

Es folgen 29 Blatt Zeichnungen

Anhängende Zeichnungen

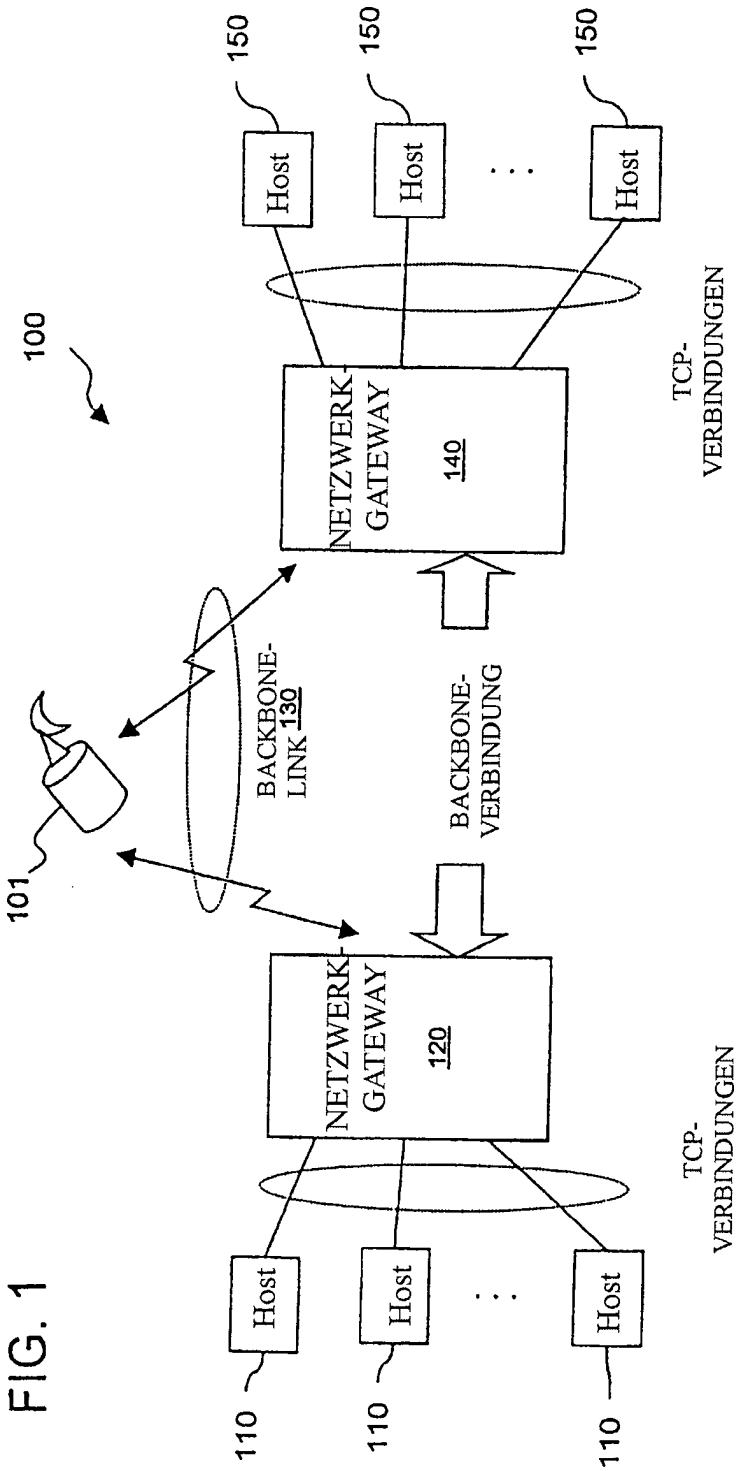


FIG. 1

FIG. 2

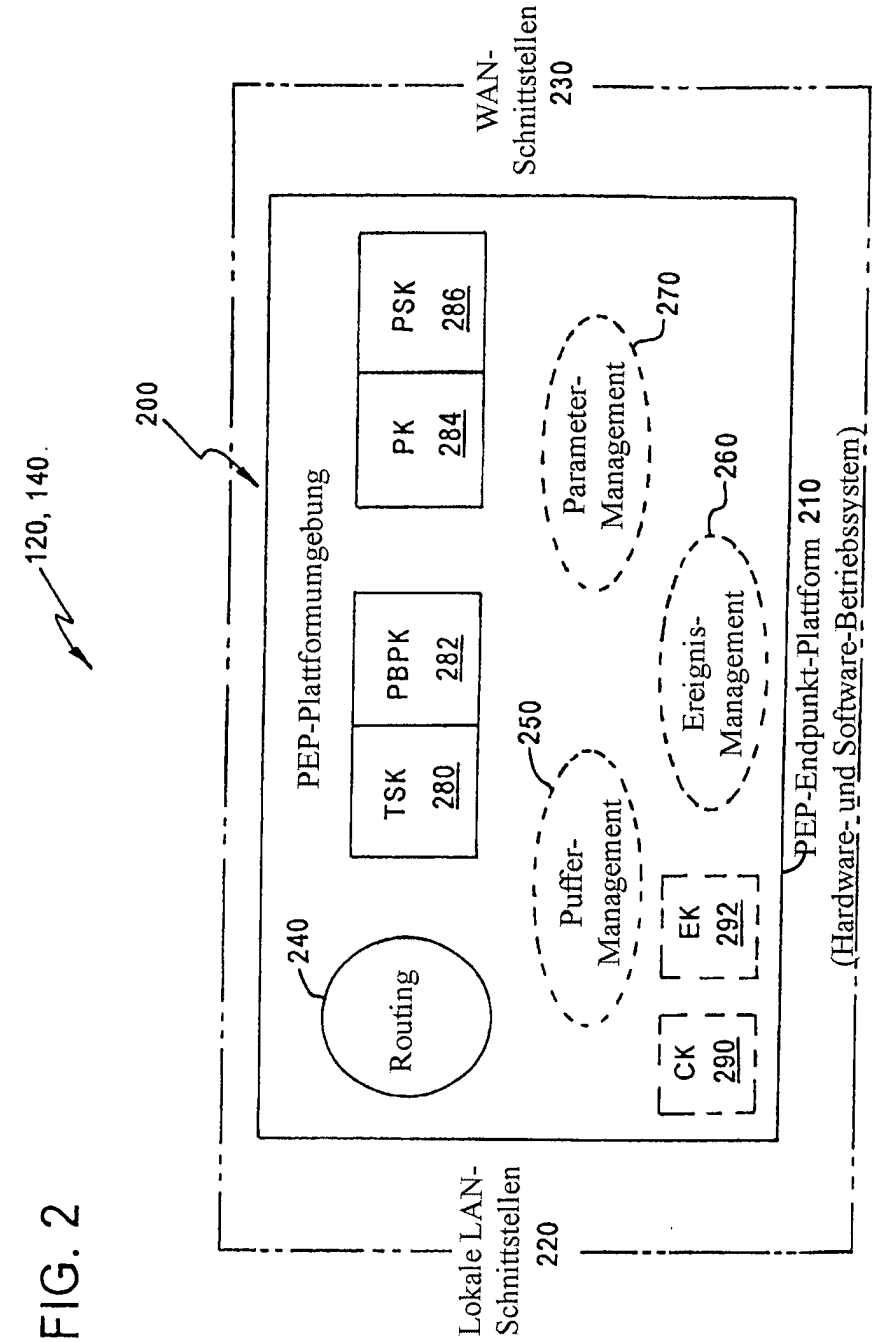

120, 140
200
WAN-Schnittstellen 230
PEP-Plattformumgebung
PK 284
PSK 286
TSK 280
PBPK 282
Parameter-Management
270
260
250
Puffer-Management
Ereignis-Management
240
Routing
CK 290
EK 292
PEP-Endpunkt-Plattform 210
(Hardware- und Software-Betriebssystem)
Lokale LAN-Schnittstellen 220

FIG. 3

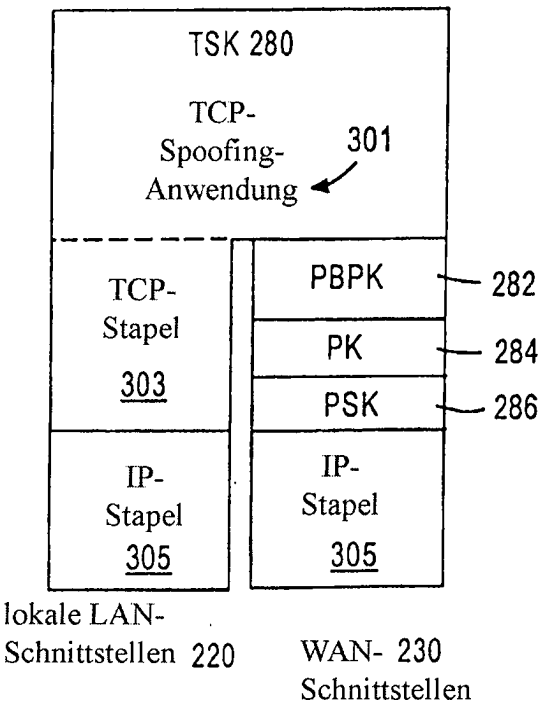

TSK 280
TCP-
Spoofing-
Anwendung
301
TCP-
Stapel
303
IP-
Stapel
305
PBPK
282
PK
284
PSK
286
IP-
Stapel
305
lokale LAN-
Schnittstellen 220
WAN- 230
Schnittstellen

FIG. 4A

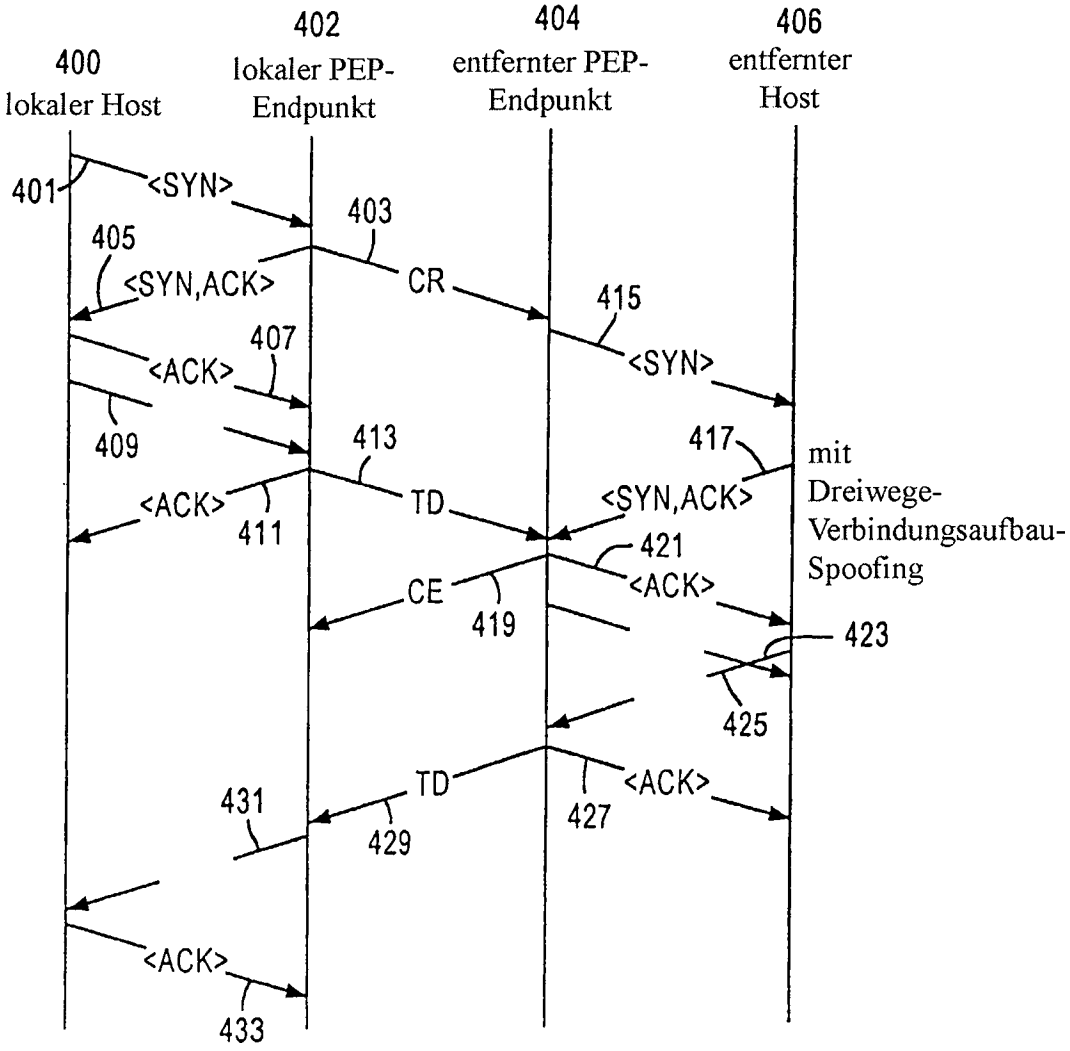

400
lokaler Host
402
lokaler PEP-Endpunkt
404
entfernter PEP-Endpunkt
406
entfernter Host
401
<SYN>
403
CR
405
<SYN,ACK>
415
<SYN>
407
<ACK>
409
413
417
mit Dreiwege-Verbindungsaufbau-Spoofing
<ACK>
411
TD
<SYN,ACK>
421
<ACK>
CE
419
423
425
TD
<ACK>
431
427
429
<ACK>
433

FIG. 4B

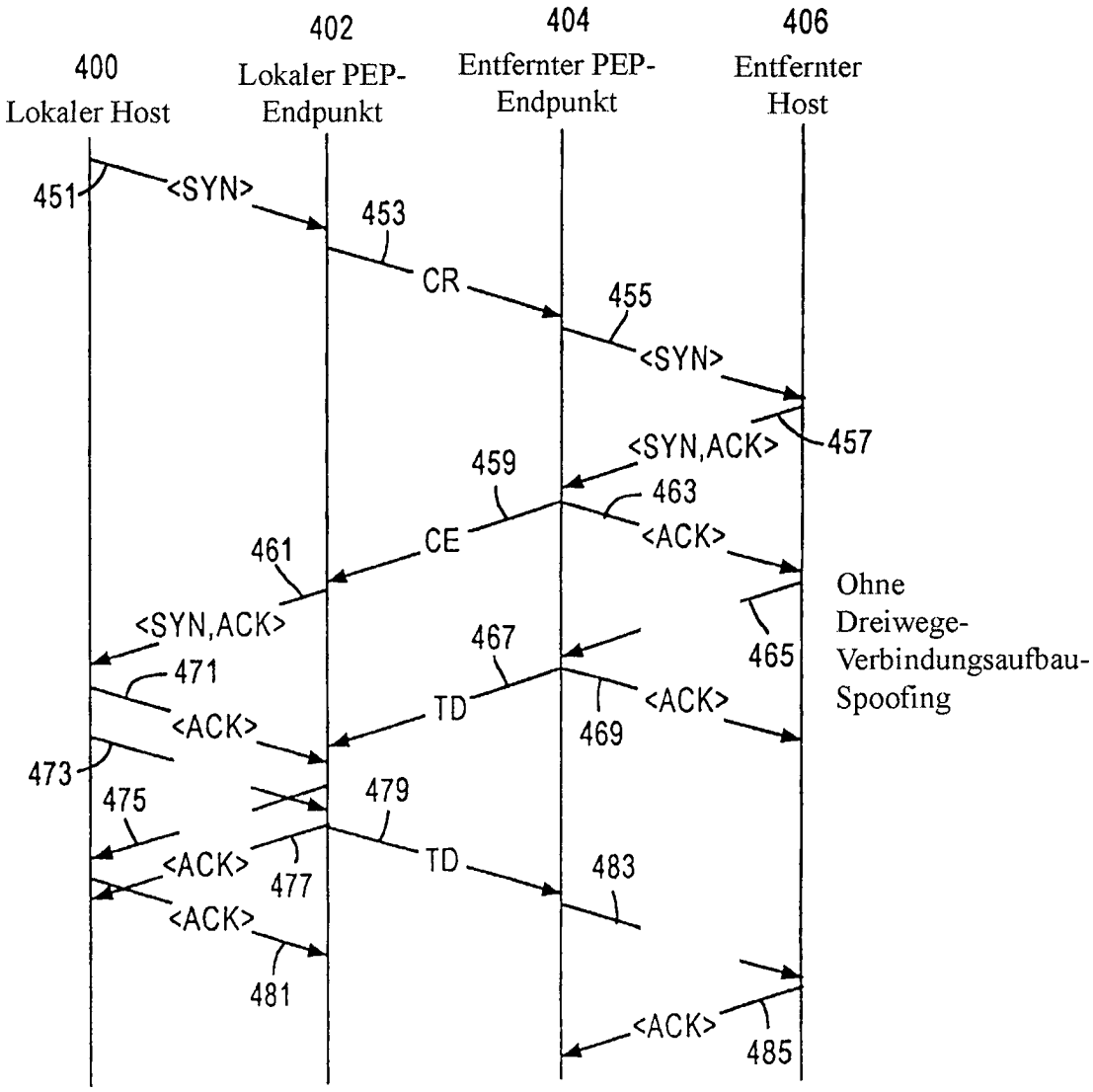

400
Lokaler Host
402
Lokaler PEP-Endpunkt
404
Entfernter PEP-Endpunkt
406
Entfernter Host
451
<SYN>
453
CR
455
<SYN>
457
<SYN,ACK>
459
CE
461
<SYN,ACK>
463
<ACK>
465
Ohne Dreiwege-Verbindungsaufbau-Spoofing
467
TD
469
<ACK>
471
<ACK>
473
475
<ACK>
477
479
TD
481
<ACK>
483
<ACK>
485

FIG. 5

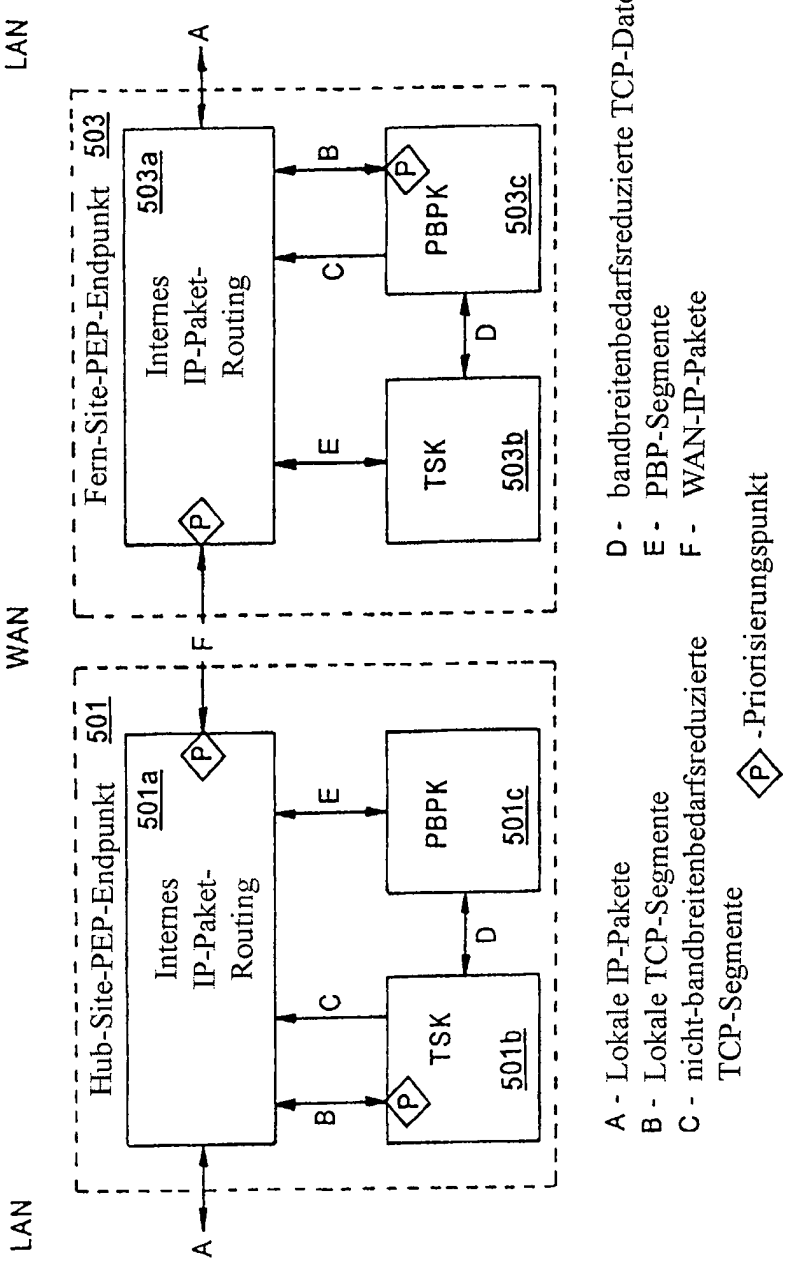

LAN
WAN
LAN
Hub-Site-PEP-Endpunkt 501
Internes IP-Paket-Routing 501a
PBPK 501c
TSK 501b
Fern-Site-PEP-Endpunkt 503
Internes IP-Paket-Routing 503a
PBPK 503c
TSK 503b
A
B
C
D
E
F
A - Lokale IP-Pakete
B - Lokale TCP-Segmente
C - nicht-bandbreitenbedarfsreduzierte TCP-Segmente
D - bandbreitenbedarfsreduzierte TCP-Daten
E - PBP-Segmente
F - WAN-IP-Pakete
P - Priorisierungspunkt

FIG. 6

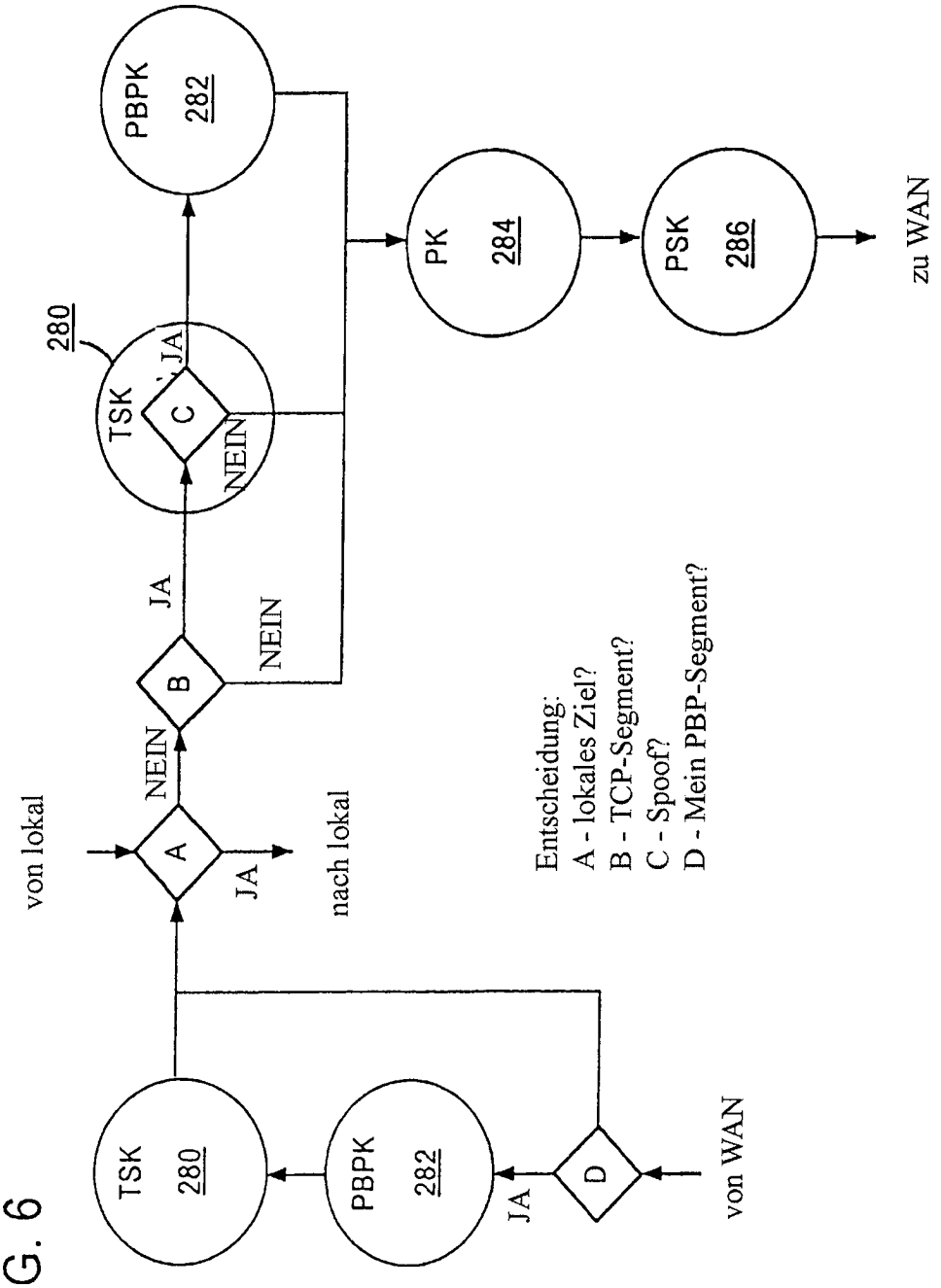

von lokal
A
NEIN
JA
nach lokal
B
JA
NEIN
280
TSK
C
JA
NEIN
PBPK
282
PK
284
PSK
286
zu WAN
Entscheidung:
A - lokales Ziel?
B - TCP-Segment?
C - Spoof?
D - Mein PBP-Segment?
TSK
280
PBPK
282
JA
D
von WAN

FIG. 7

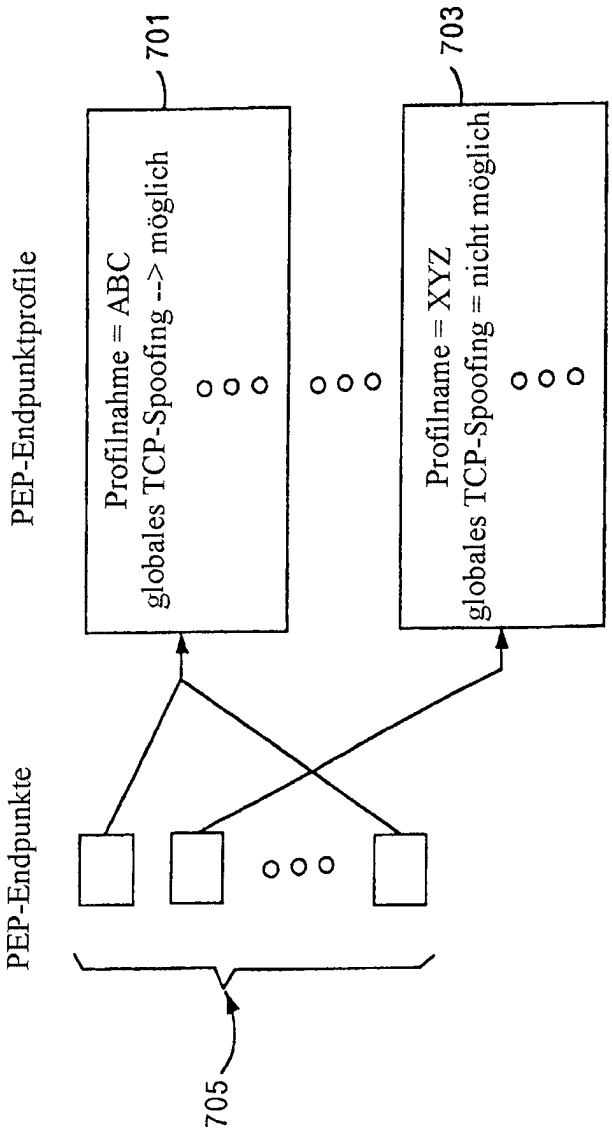

PEP-Endpunktprofile
701
Profilnahme = ABC
globales TCP-Spoofing --> möglich
703
Profilname = XYZ
globales TCP-Spoofing = nicht möglich
PEP-Endpunkte
705

FIG. 8

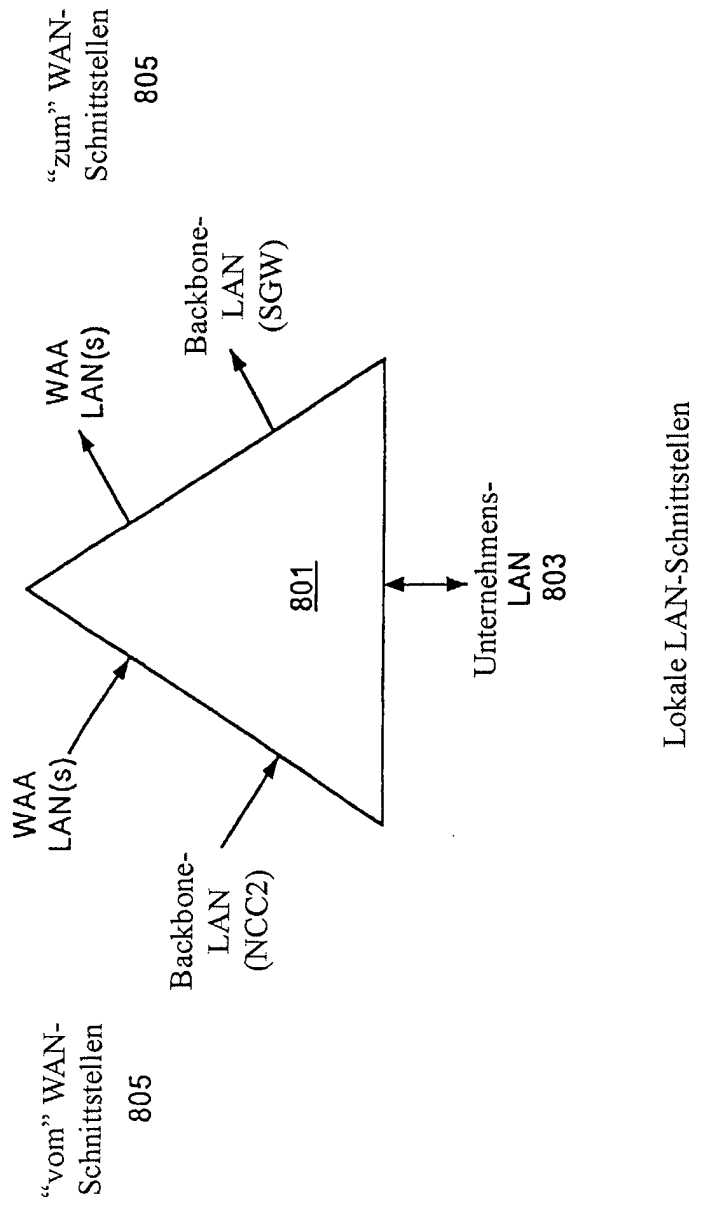

"zum" WAN-Schnittstellen
805
WAA LAN(s)
Backbone-LAN (SGW)
801
Unternehmens-LAN 803
WAA LAN(s)
Backbone-LAN (NCC2)
"vom" WAN-Schnittstellen
805
Lokale LAN-Schnittstellen

FIG. 9

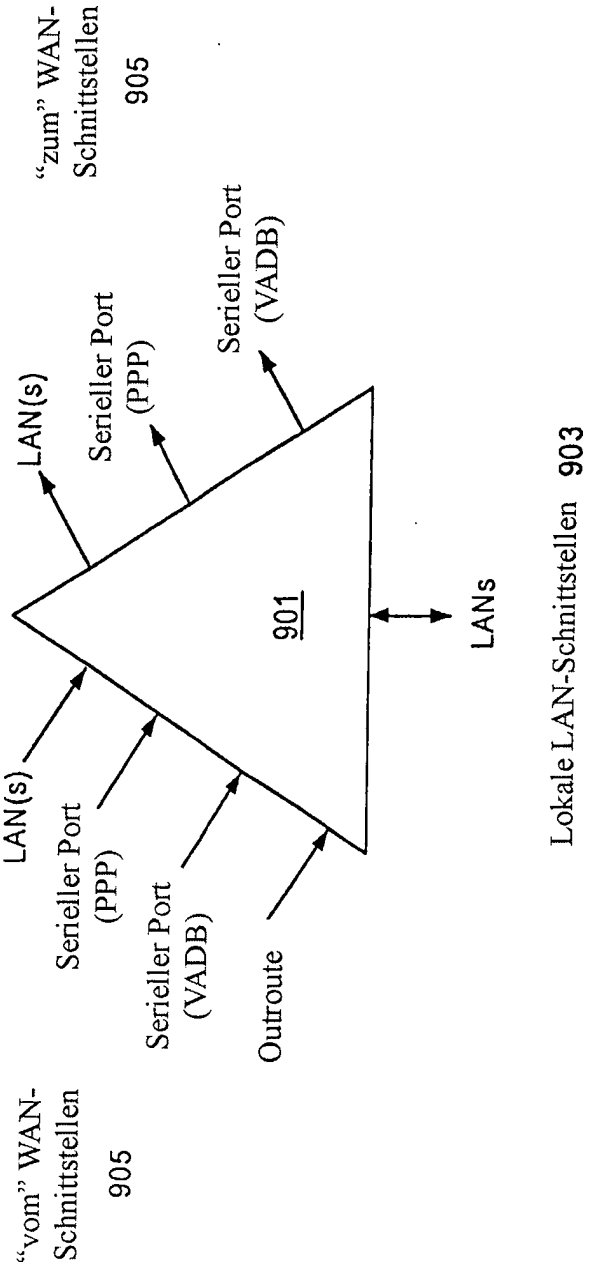

"vom" WAN-Schnittstellen
905
LAN(s)
Serieller Port (PPP)
Serieller Port (VADB)
Outroute
901
LANs
Lokale LAN-Schnittstellen 903
LAN(s)
Serieller Port (PPP)
Serieller Port (VADB)
"zum" WAN-Schnittstellen
905

FIG. 10

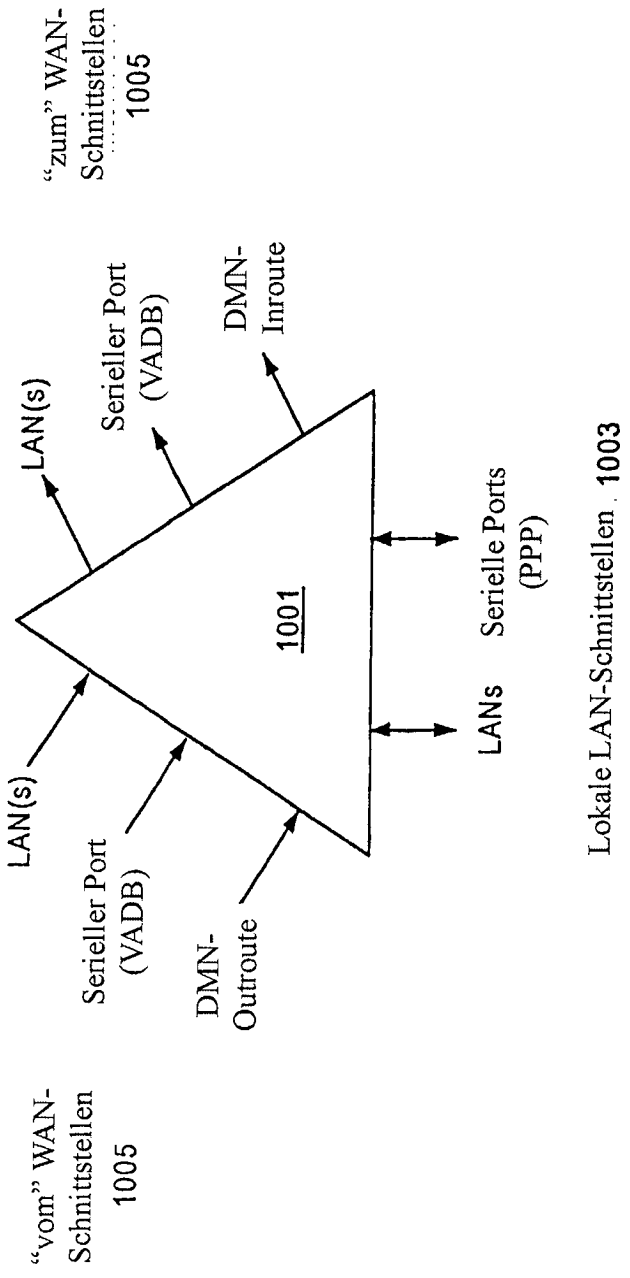

"vom" WAN-Schnittstellen 1005
"zum" WAN-Schnittstellen 1005
LAN(s)
Serieller Port (VADB)
DMN-Outroute
LAN(s)
Serieller Port (VADB)
DMN-Inroute
1001
LANs
Serielle Ports (PPP)
Lokale LAN-Schnittstellen 1003

FIG. 11

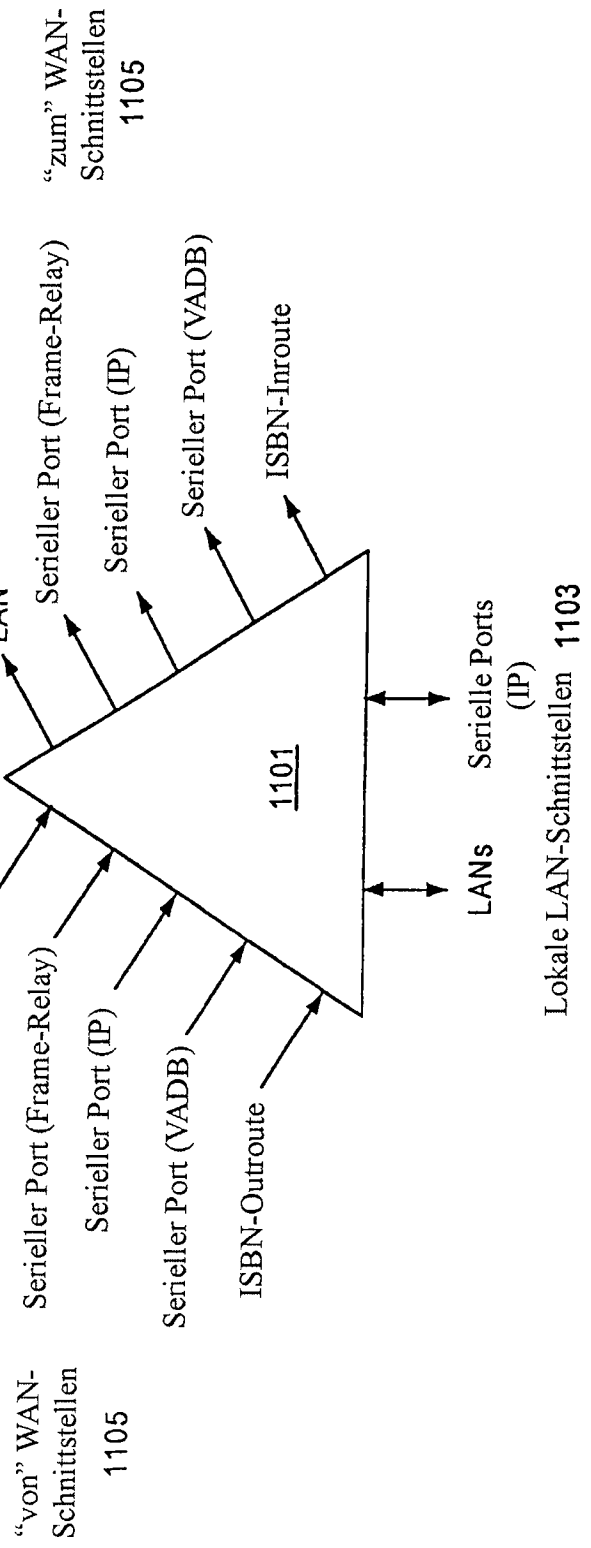

"zum" WAN-Schnittstellen 1105
LAN
Serieller Port (Frame-Relay)
Serieller Port (IP)
Serieller Port (VADB)
ISBN-Inroute
1101
LAN
Serieller Port (Frame-Relay)
Serieller Port (IP)
Serieller Port (VADB)
ISBN-Outroute
"von" WAN-Schnittstellen 1105
LANs
Serielle Ports (IP)
Lokale LAN-Schnittstellen 1103

FIG. 12

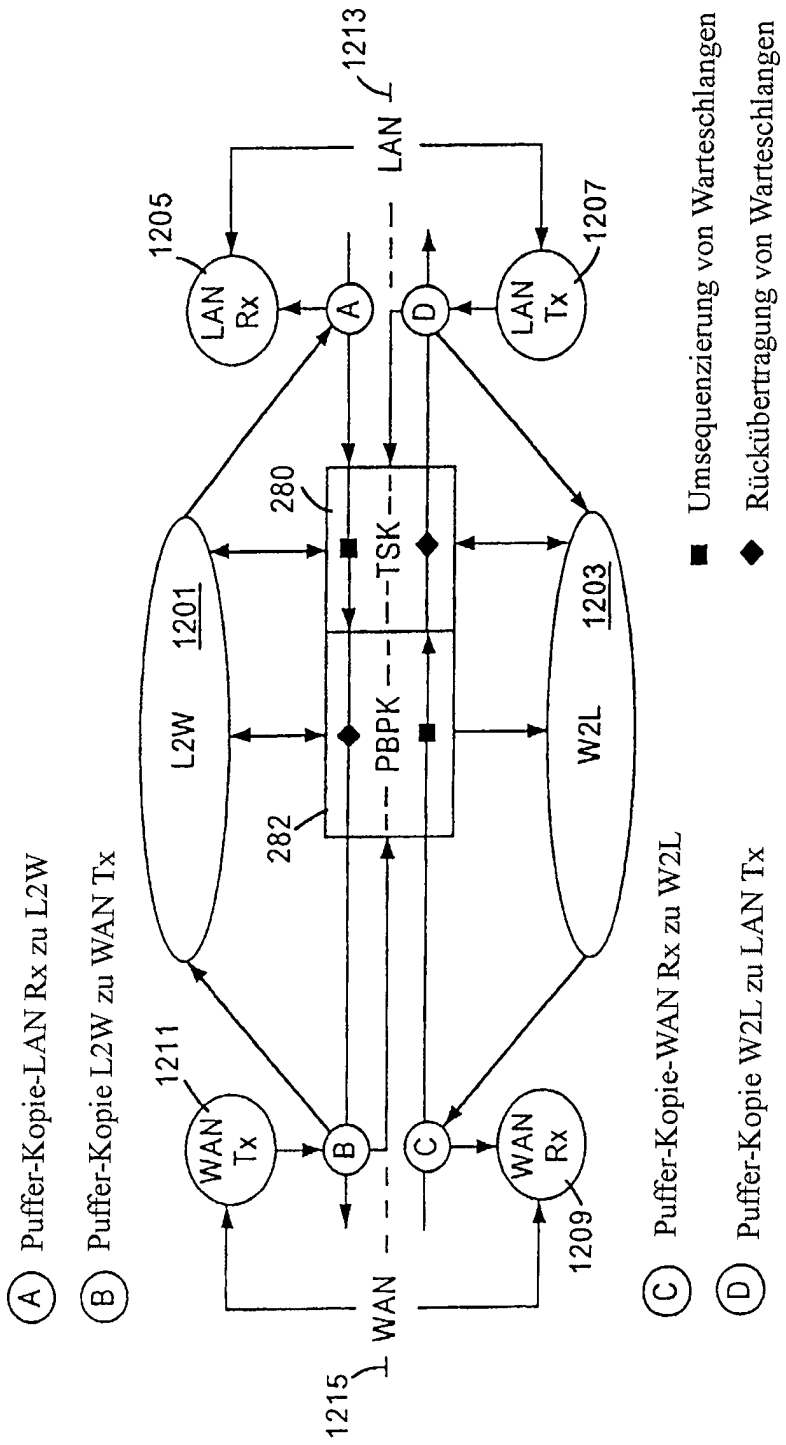

A Puffer-Kopie-LAN Rx zu L2W
B Puffer-Kopie L2W zu WAN Tx
C Puffer-Kopie-WAN Rx zu W2L
D Puffer-Kopie W2L zu LAN Tx
Umsequenzierung von Warteschlangen
Rückübertragung von Warteschlangen
LAN
WAN
LAN Rx
LAN Tx
WAN Tx
WAN Rx
L2W
W2L
TSK
PBPK
1201
1203
1205
1207
1209
1211
1213
1215
280
282

FIG. 13

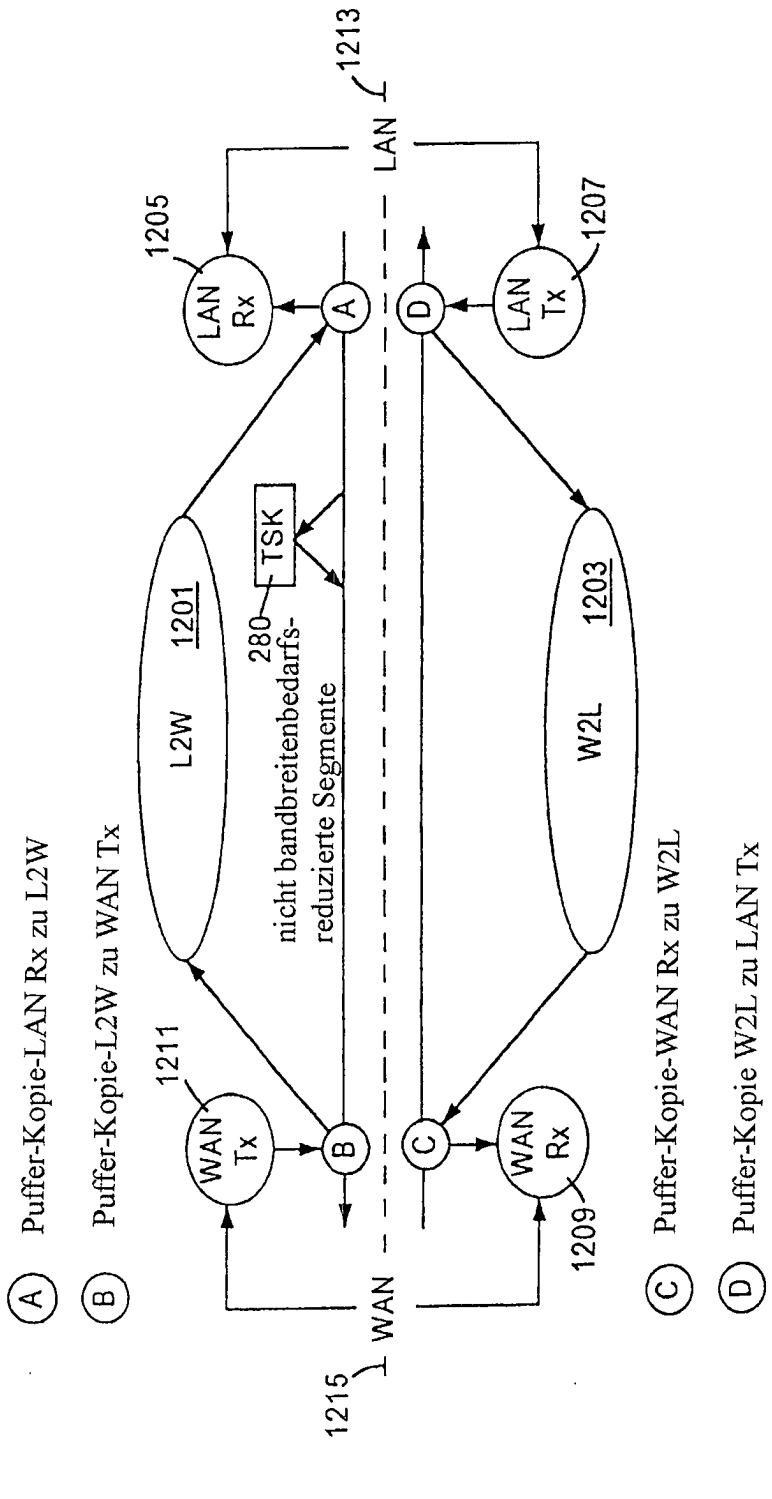

1213
LAN
1205
LAN Rx
A
D
LAN Tx
1207
TSK
280
nicht bandbreitenbedarfs-reduzierte Segmente
L2W 1201
W2L 1203
1211
WAN Tx
B
C
WAN Rx
1209
WAN
1215
A Puffer-Kopie-LAN Rx zu L2W
B Puffer-Kopie-L2W zu WAN Tx
C Puffer-Kopie-WAN Rx zu W2L
D Puffer-Kopie W2L zu LAN Tx

FIG. 14

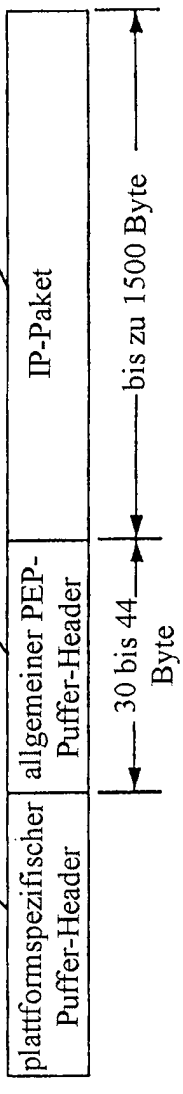

1400
1401
plattformspezifischer Puffer-Header
1403
allgemeiner PEP-Puffer-Header
1405
IP-Paket
30 bis 44 Byte
bis zu 1500 Byte

FIG. 15

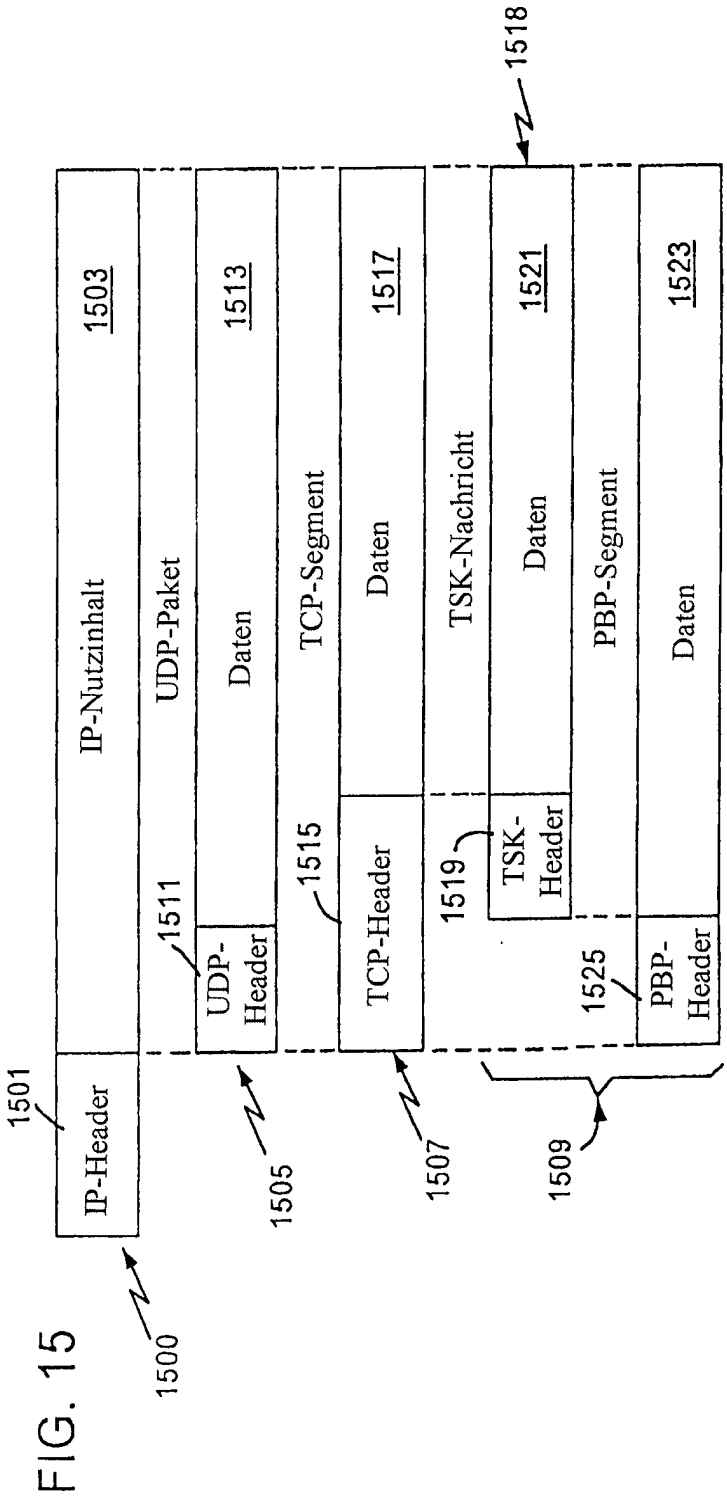

1500
1501
IP-Header
IP-Nutzinhalt 1503
UDP-Paket
1505
UDP-Header
1511
Daten 1513
TCP-Segment
1507
TCP-Header
1515
Daten 1517
TSK-Nachricht
1509
TSK-Header
1519
Daten 1521
1518
PBP-Segment
1525
PBP-Header
Daten 1523

FIG. 16

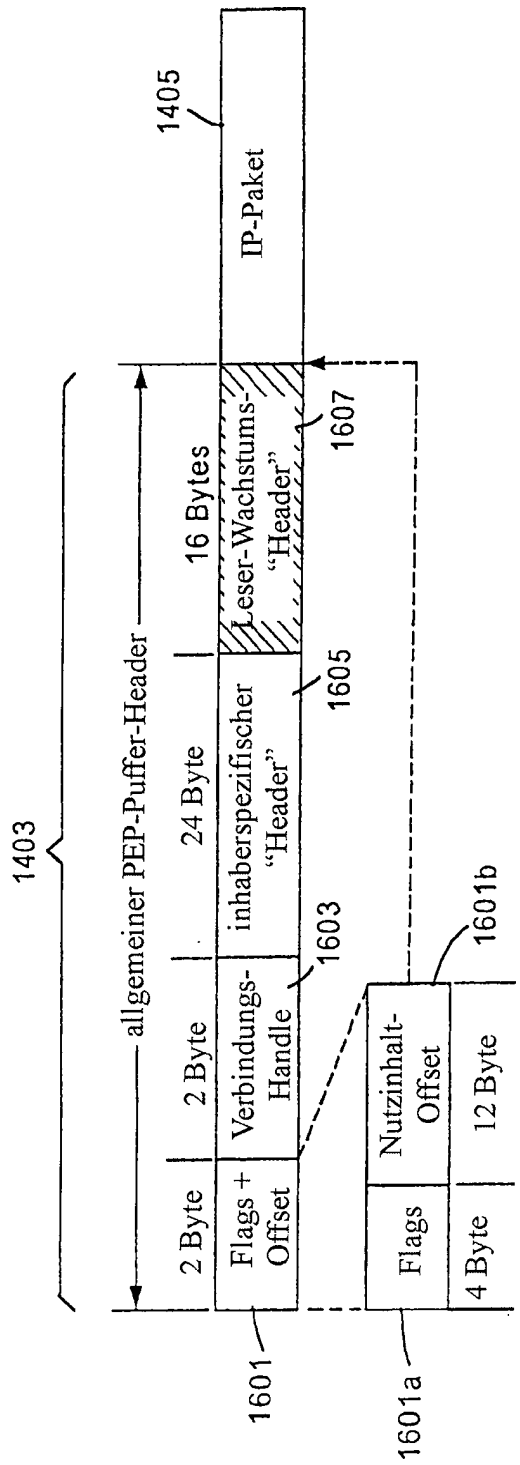

1403
allgemeiner PEP-Puffer-Header
2 Byte
Flags + Offset
2 Byte
Verbindungs-Handle
24 Byte
inhaberspezifischer "Header"
16 Bytes
Leser-Wachstums-"Header"
IP-Paket
1405
1607
1605
1603
1601
1601a
Flags
4 Byte
Nutzinhalt-Offset
12 Byte
1601b

FIG. 17

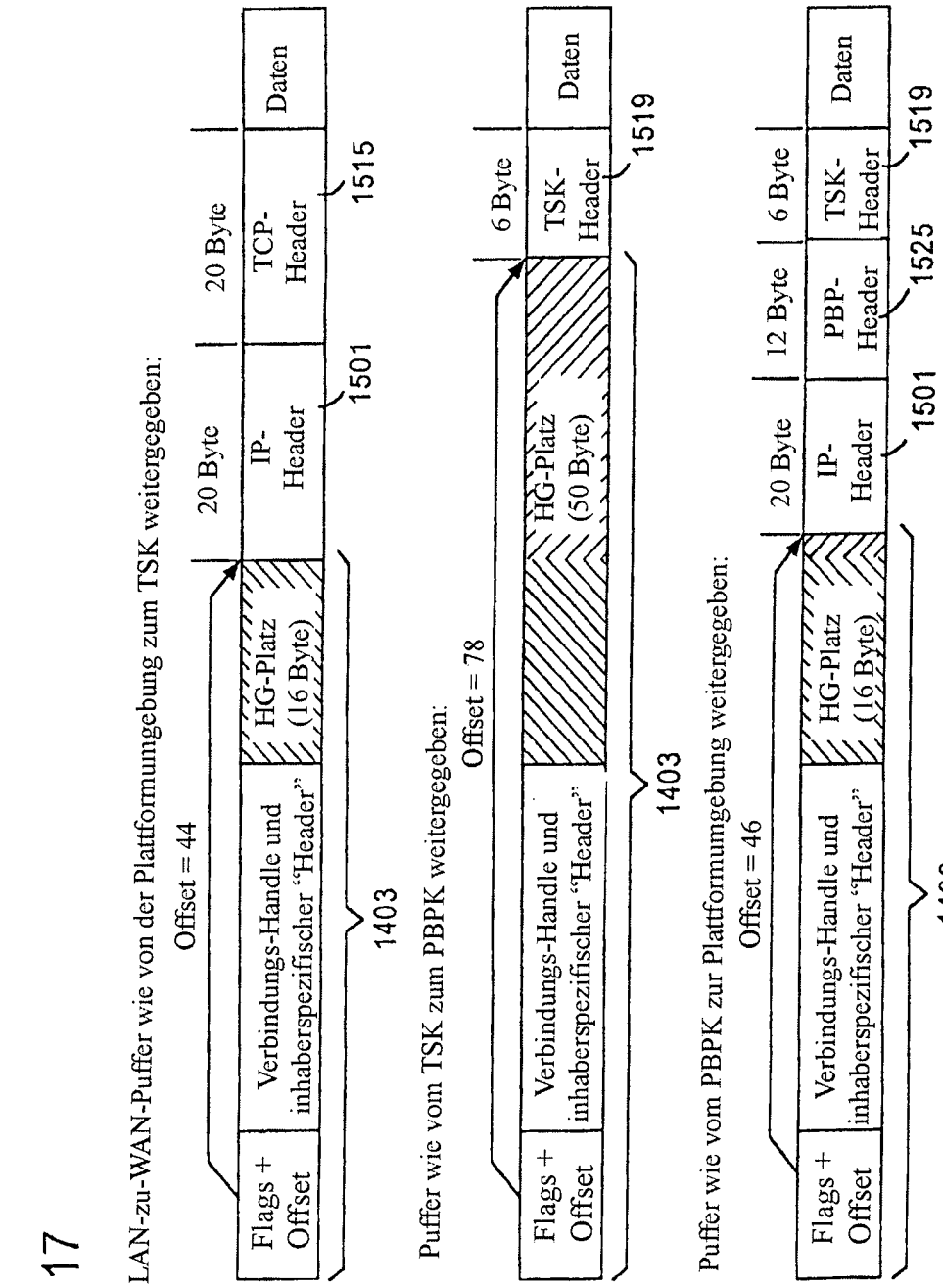

LAN-zu-WAN-Puffer wie von der Plattformumgebung zum TSK weitergegeben:
Offset = 44
20 Byte
20 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
IP-Header
TCP-Header
Daten
1501
1515
1403
Puffer wie vom TSK zum PBPK weitergegeben:
Offset = 78
6 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (50 Byte)
TSK-Header
Daten
1519
1403
Puffer wie vom PBPK zur Plattformumgebung weitergegeben:
Offset = 46
20 Byte
12 Byte
6 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
IP-Header
PBP-Header
TSK-Header
Daten
1501
1525
1519
1403

FIG. 18

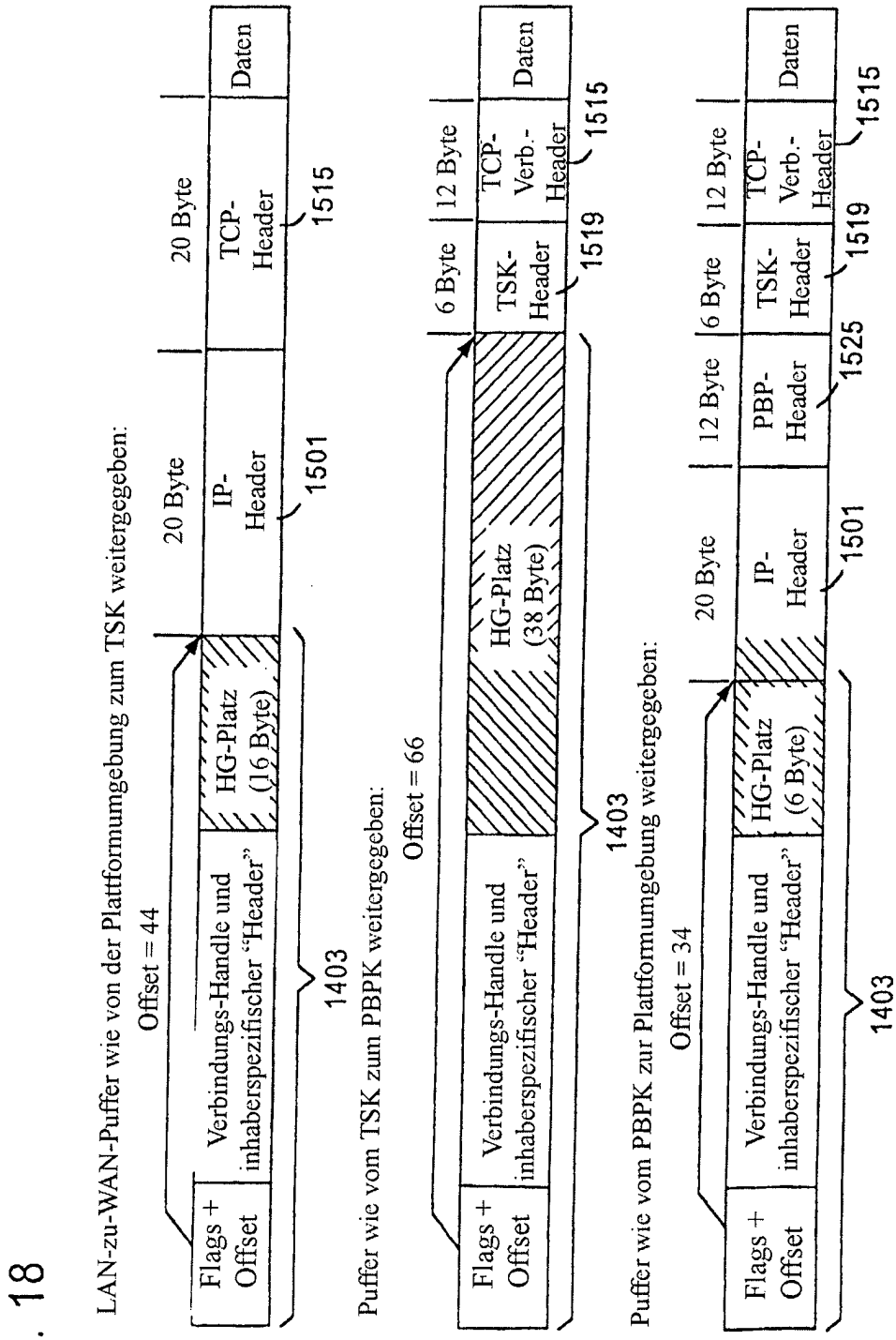

LAN-zu-WAN-Puffer wie von der Plattformumgebung zum TSK weitergegeben:
Offset = 44
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
20 Byte
IP-Header
20 Byte
TCP-Header
Daten
1501
1515
1403
Puffer wie vom TSK zum PBPK weitergegeben:
Offset = 66
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (38 Byte)
6 Byte
TSK-Header
12 Byte
TCP-Verb.-Header
Daten
1519
1515
1403
Puffer wie vom PBPK zur Plattformumgebung weitergegeben:
Offset = 34
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (6 Byte)
20 Byte
IP-Header
12 Byte
PBP-Header
6 Byte
TSK-Header
12 Byte
TCP-Verb.-Header
Daten
1501
1525
1519
1515
1403

FIG. 19

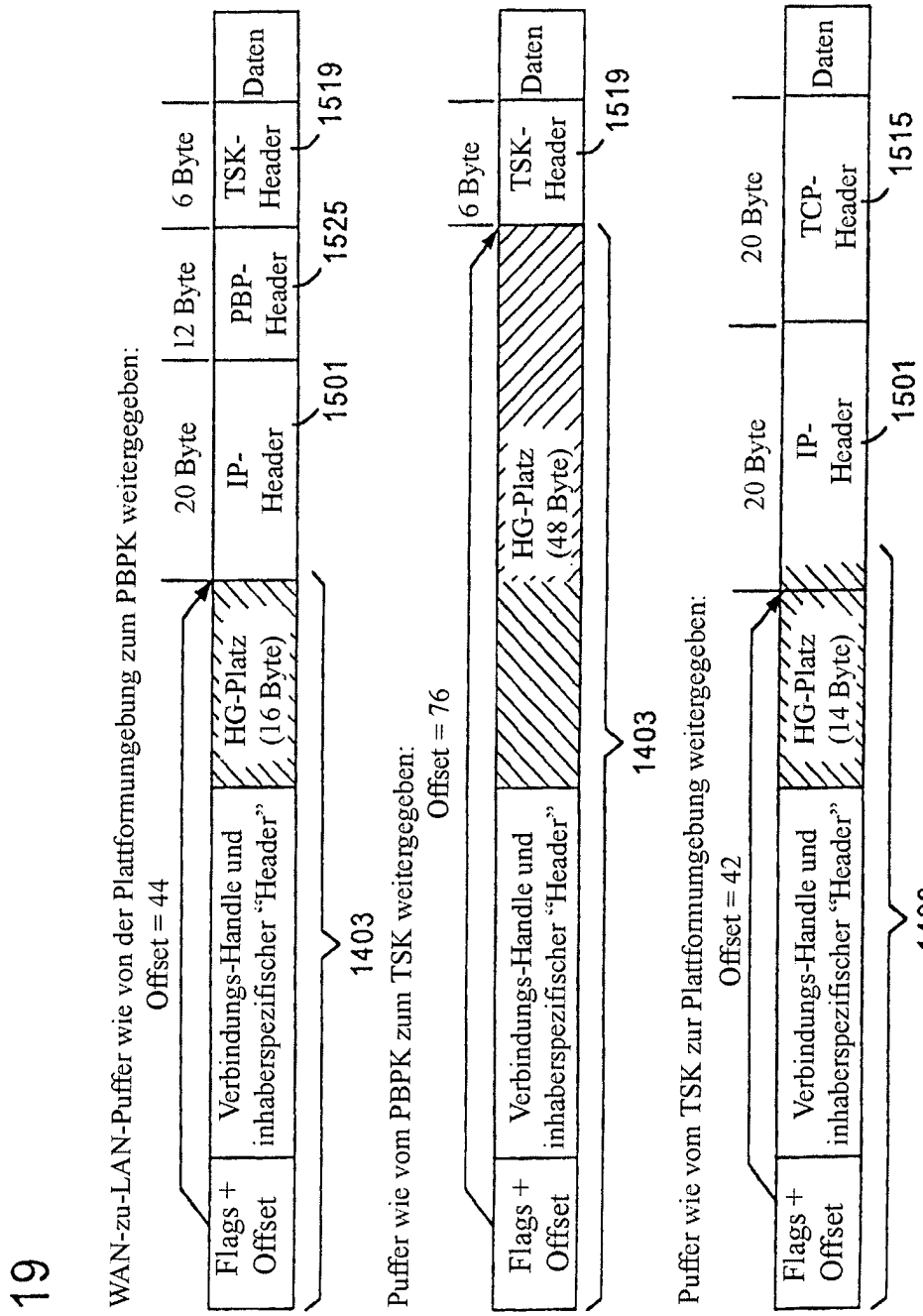

WAN-zu-LAN-Puffer wie von der Plattformumgebung zum PBPK weitergegeben:
Offset = 44
20 Byte
12 Byte
6 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
IP-Header
PBP-Header
TSK-Header
Daten
1501
1525
1519
1403
Puffer wie vom PBPK zum TSK weitergegeben:
Offset = 76
6 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (48 Byte)
TSK-Header
Daten
1519
1403
Puffer wie vom TSK zur Plattformumgebung weitergegeben:
Offset = 42
20 Byte
20 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (14 Byte)
IP-Header
TCP-Header
Daten
1501
1515
1403

FIG. 20

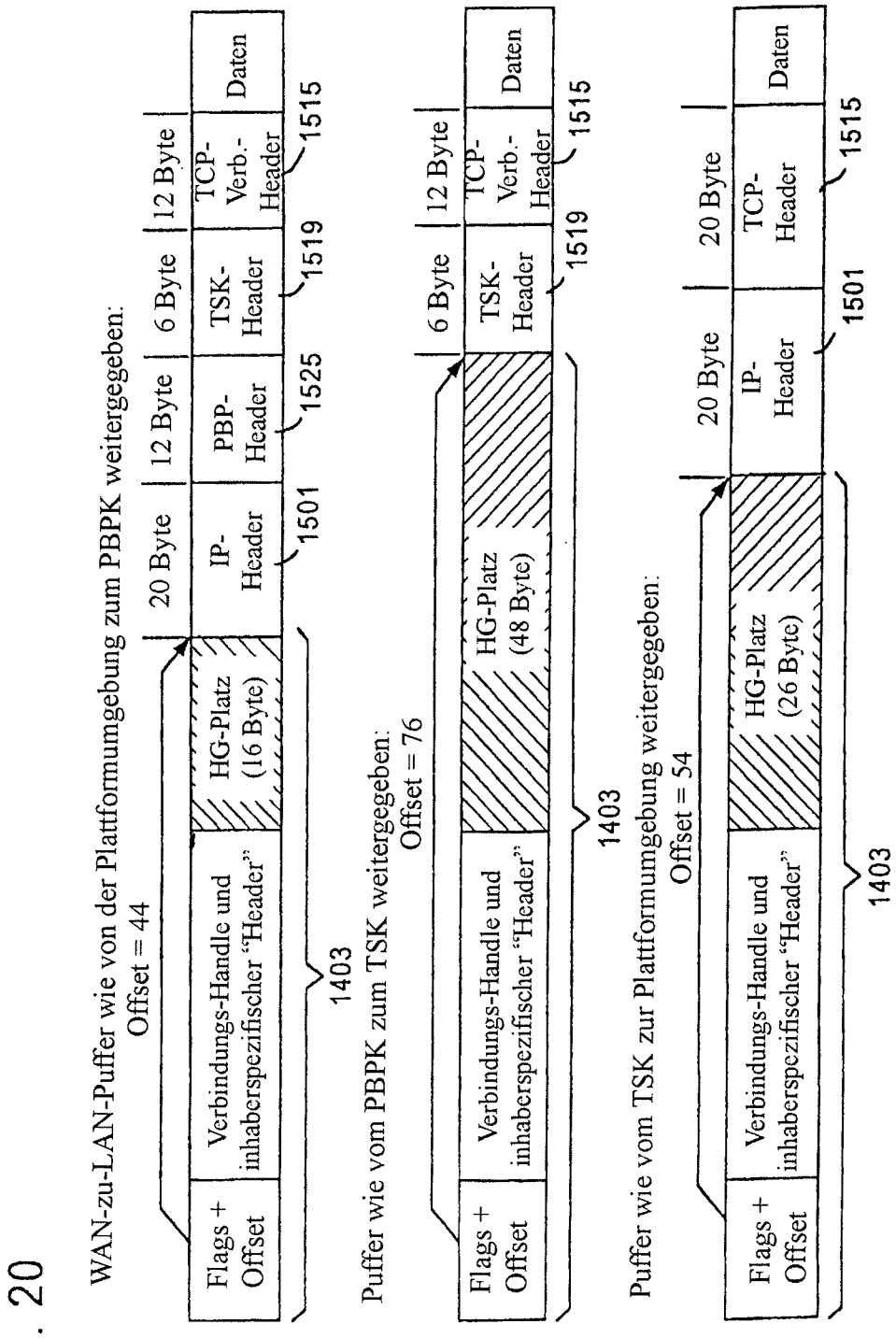

WAN-zu-LAN-Puffer wie von der Plattformumgebung zum PBPK weitergegeben:
Offset = 44
20 Byte
12 Byte
6 Byte
12 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
IP-Header
PBP-Header
TSK-Header
TCP-Verb.-Header
Daten
1501
1525
1519
1515
1403
Puffer wie vom PBPK zum TSK weitergegeben:
Offset = 76
6 Byte
12 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (48 Byte)
TSK-Header
TCP-Verb.-Header
Daten
1519
1515
1403
Puffer wie vom TSK zur Plattformumgebung weitergegeben:
Offset = 54
20 Byte
20 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (26 Byte)
IP-Header
TCP-Header
Daten
1501
1515
1403

FIG. 21

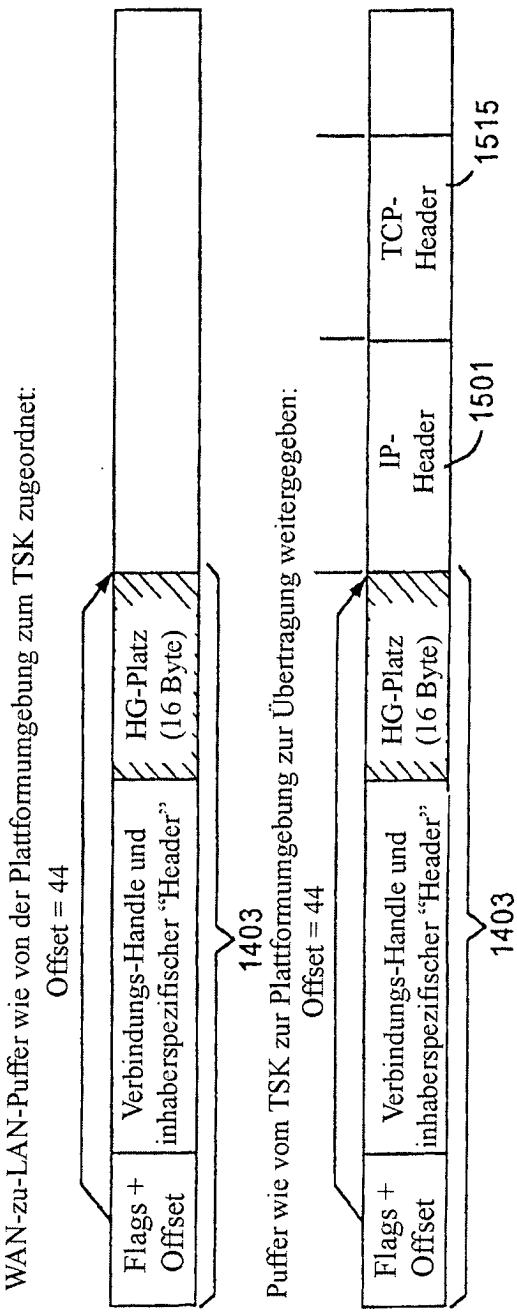

WAN-zu-LAN-Puffer wie von der Plattformumgebung zum TSK zugeordnet:
Offset = 44
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
1403
Puffer wie vom TSK zur Plattformumgebung zur Übertragung weitergegeben:
Offset = 44
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
IP-Header
1501
TCP-Header
1515
1403

FIG. 22

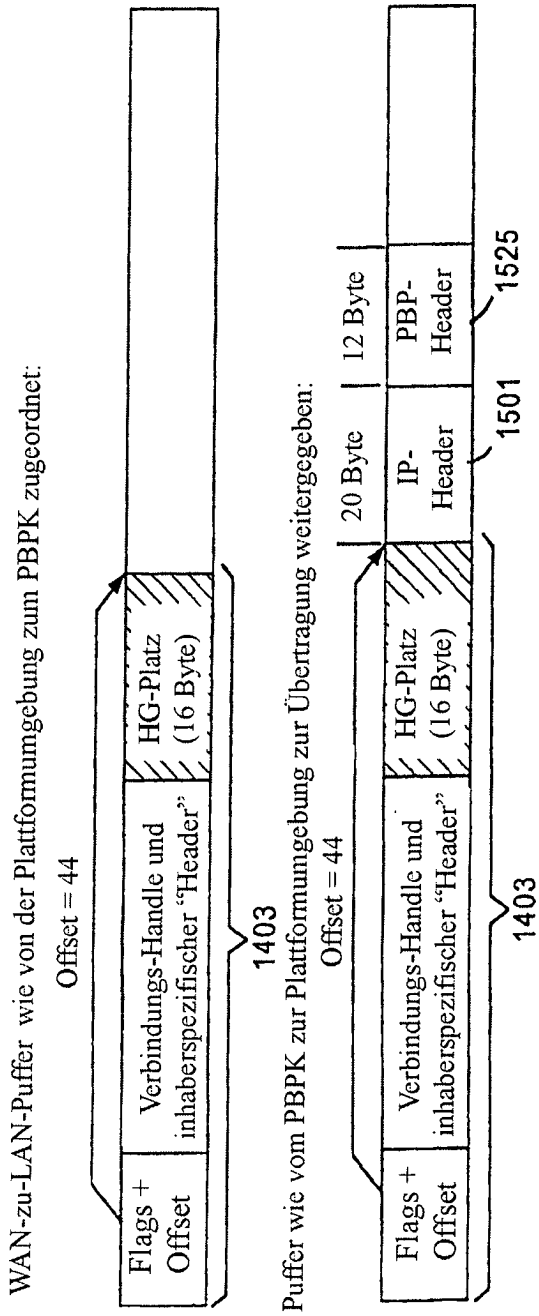

WAN-zu-LAN-Puffer wie von der Plattformumgebung zum PBPK zugeordnet:
Offset = 44
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
1403
Puffer wie vom PBPK zur Plattformumgebung zur Übertragung weitergegeben:
Offset = 44
20 Byte
12 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
IP-Header
1501
PBP-Header
1525
1403

FIG. 23

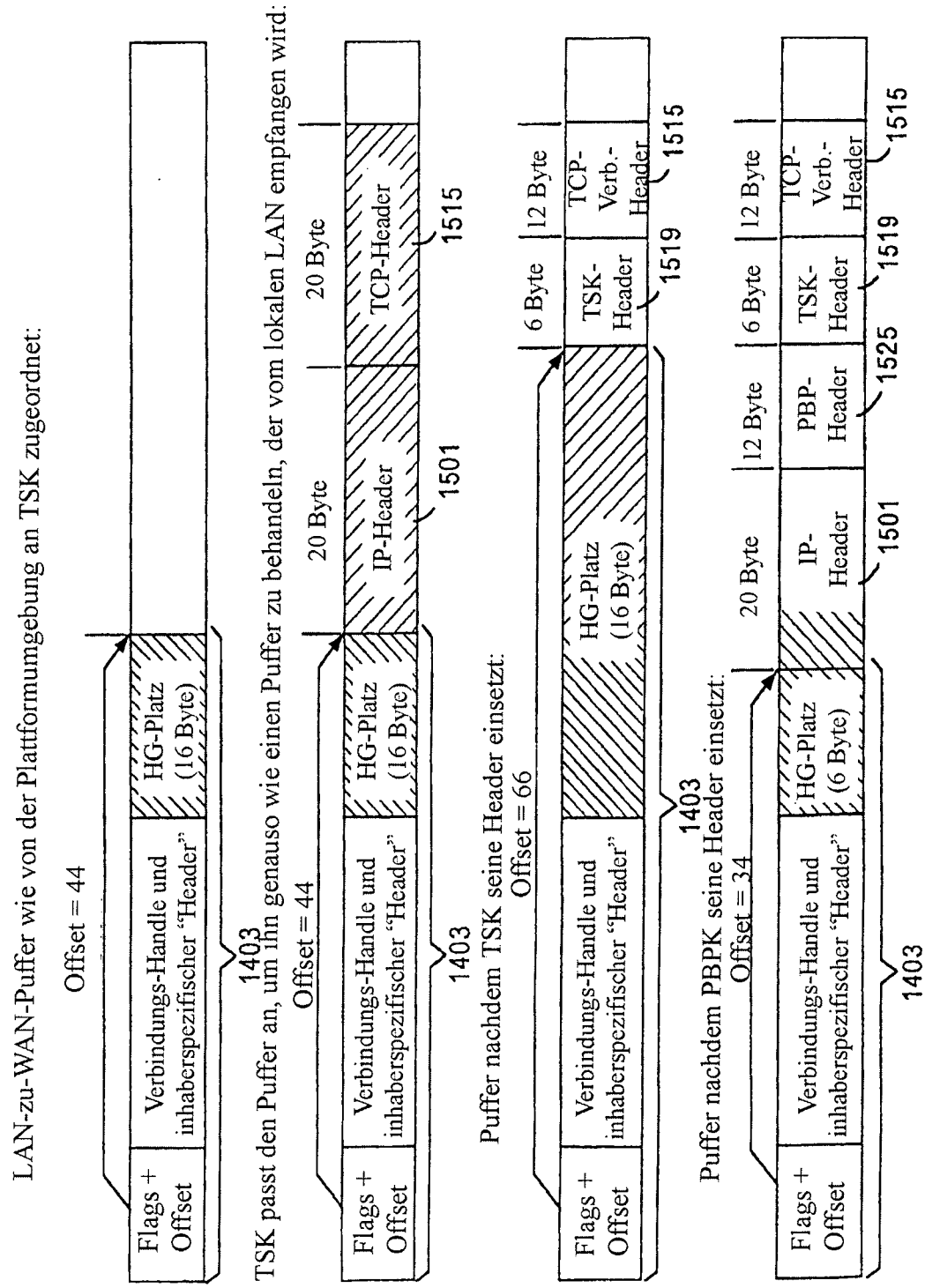

LAN-zu-WAN-Puffer wie von der Plattformumgebung an TSK zugeordnet:
Offset = 44
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
1403
TSK passt den Puffer an, um ihn genauso wie einen Puffer zu behandeln, der vom lokalen LAN empfangen wird:
Offset = 44
20 Byte
20 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
IP-Header
TCP-Header
1501
1515
1403
Puffer nachdem TSK seine Header einsetzt:
Offset = 66
6 Byte
12 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
TSK-Header
TCP-Verb.-Header
1519
1515
1403
Puffer nachdem PBPK seine Header einsetzt:
Offset = 34
20 Byte
12 Byte
6 Byte
12 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (6 Byte)
IP-Header
PBP-Header
TSK-Header
TCP-Verb.-Header
1501
1525
1519
1515
1403

FIG. 24

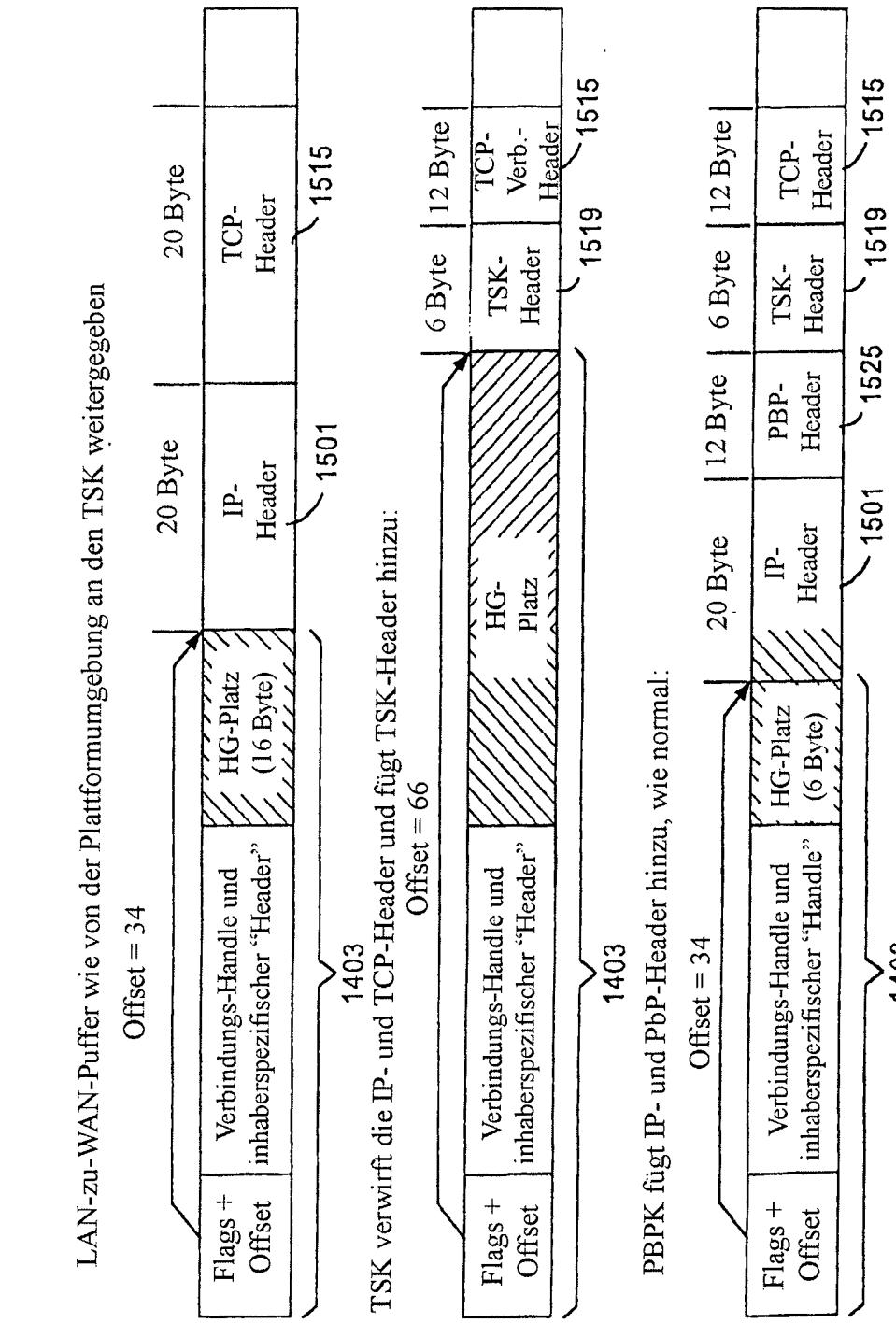
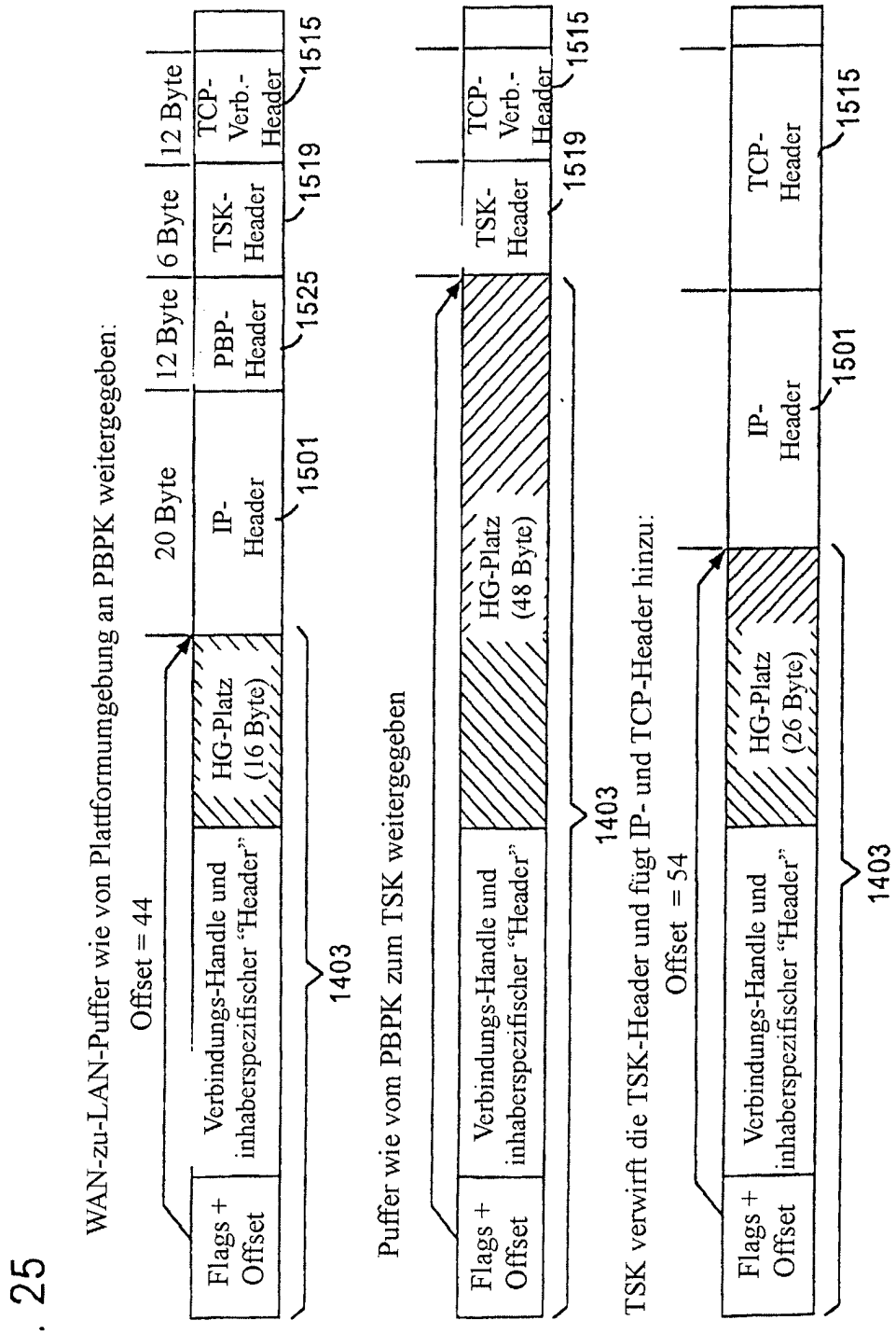

LAN-zu-WAN-Puffer wie von der Plattformumgebung an den TSK weitergegeben
Offset = 34
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
20 Byte
IP-Header
20 Byte
TCP-Header
1501
1515
1403
TSK verwirft die IP- und TCP-Header und fügt TSK-Header hinzu:
Offset = 66
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz
6 Byte
TSK-Header
12 Byte
TCP-Verb.-Header
1519
1515
1403
PBPK fügt IP- und PbP-Header hinzu, wie normal:
Offset = 34
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Handle"
HG-Platz (6 Byte)
20 Byte
IP-Header
12 Byte
PBP-Header
6 Byte
TSK-Header
12 Byte
TCP-Header
1501
1525
1519
1515
1403

FIG. 25


WAN-zu-LAN-Puffer wie von Plattformumgebung an PBPK weitergegeben:
Offset = 44
20 Byte
12 Byte
6 Byte
12 Byte
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (16 Byte)
IP-Header
PBP-Header
TSK-Header
TCP-Verb.-Header
1501
1525
1519
1515
1403
Puffer wie vom PBPK zum TSK weitergegeben
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (48 Byte)
TSK-Header
TCP-Verb.-Header
1519
1515
1403
TSK verwirft die TSK-Header und fügt IP- und TCP-Header hinzu:
Offset = 54
Flags + Offset
Verbindungs-Handle und inhaberspezifischer "Header"
HG-Platz (26 Byte)
IP-Header
TCP-Header
1501
1515
1403

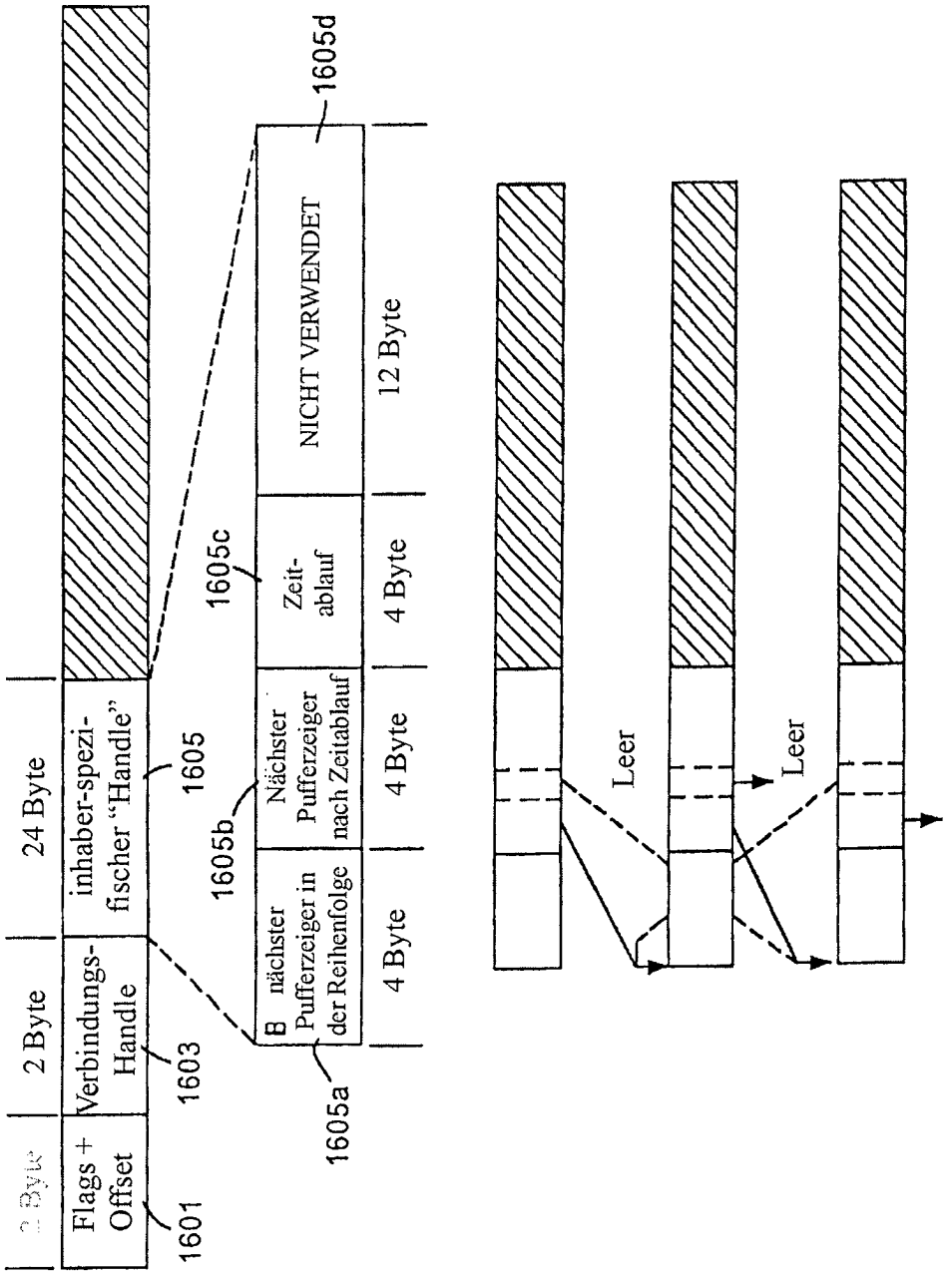

2 Byte
2 Byte
24 Byte
Flags + Offset
Verbindungs-Handle
inhaber-spezi-fischer "Handle"
1601
1603
1605
1605a
1605b
1605c
1605d
nächster Pufferzeiger in der Reihenfolge
Nächster Pufferzeiger nach Zeitablauf
Zeit-ablauf
NICHT VERWENDET
4 Byte
4 Byte
4 Byte
12 Byte
Leer
Leer

FIG. 26

FIG. 27

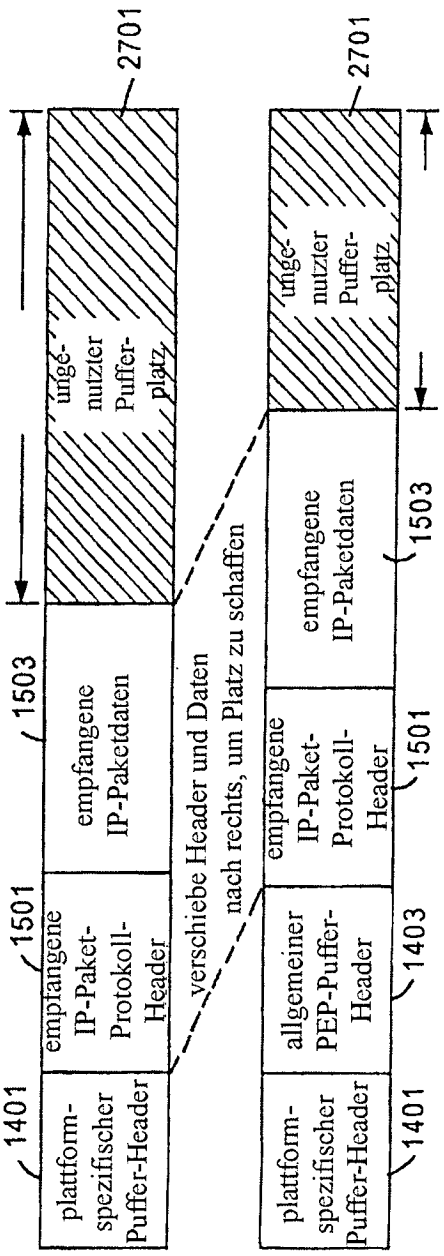
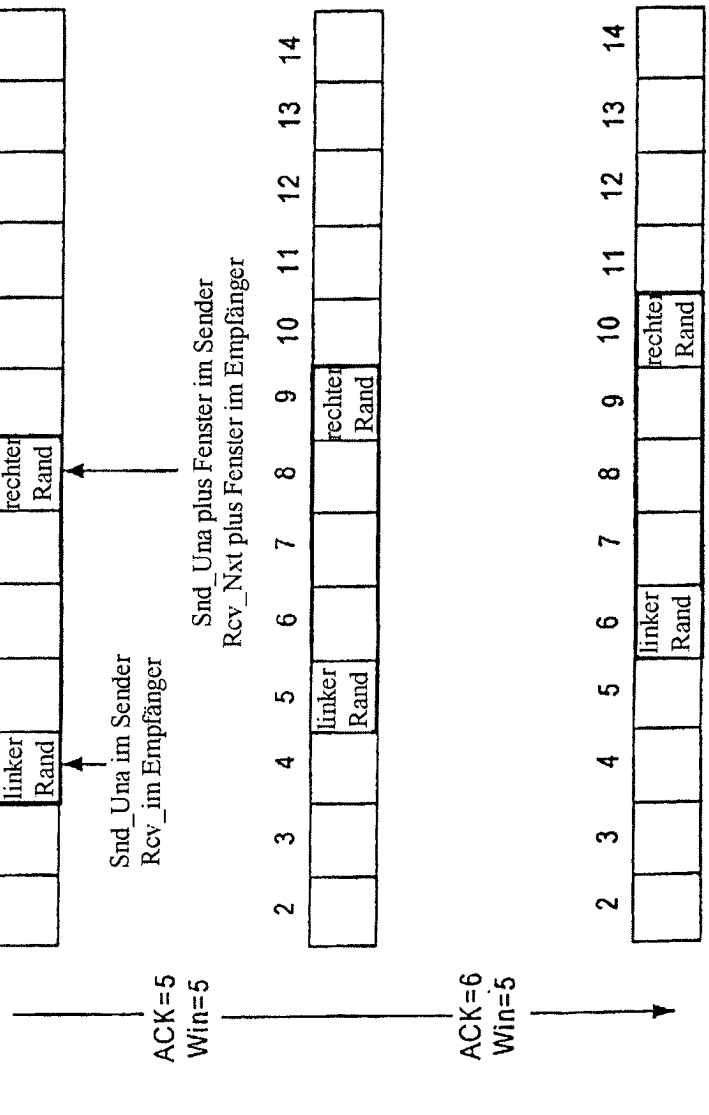

ursprünglicher Puffer
plattform-spezifischer Puffer-Header
1401
empfangene IP-Paket-Protokoll-Header
1501
empfangene IP-Paketdaten
1503
ungenutzter Puffer-platz
2701
verschiebe Header und Daten nach rechts, um Platz zu schaffen
modifizierter ursprünglicher Puffer
plattform-spezifischer Puffer-Header
1401
allgemeiner PEP-Puffer-Header
1403
empfangene IP-Paket-Protokoll-Header
1501
empfangene IP-Paketdaten
1503
ungenutzter Puffer-platz
2701

FIG. 28

modifizierter ursprünglicher Puffer
plattform-spezifischer Puffer-Header
1401
empfangene IP-Paketdaten
1503
ungenutzter Puffer-platz
2701
verschiebe Daten nach Header-Kopie zur Vorderseite des ursprünglichen Puffers
ursprünglicher Puffer
1401
empfangene IP-Paket-Protokoll-Header
1501
empfangene IP-Paketdaten
1503
kopiere Header in neuen Puffer und verknüpfe mit Vorderseite des ursprünglichen Puffers
neuer Puffer
plattform-spezifischer Puffer-Header
1401
allgemeiner PEP-Puffer-Header
1403
empfangene IP-Paket-Protokoll-Header
1501
ungenutzter Puffer-platz
1701

FIG. 29


2901
2 3 4 5 6 7 8 9 10 11 12 13 14
linker Rand
rechter Rand
Snd_Una im Sender
Rcv_im Empfänger
Snd_Una plus Fenster im Sender
Rcv_Nxt plus Fenster im Empfänger
ACK=5
Win=5
ACK=6
Win=5

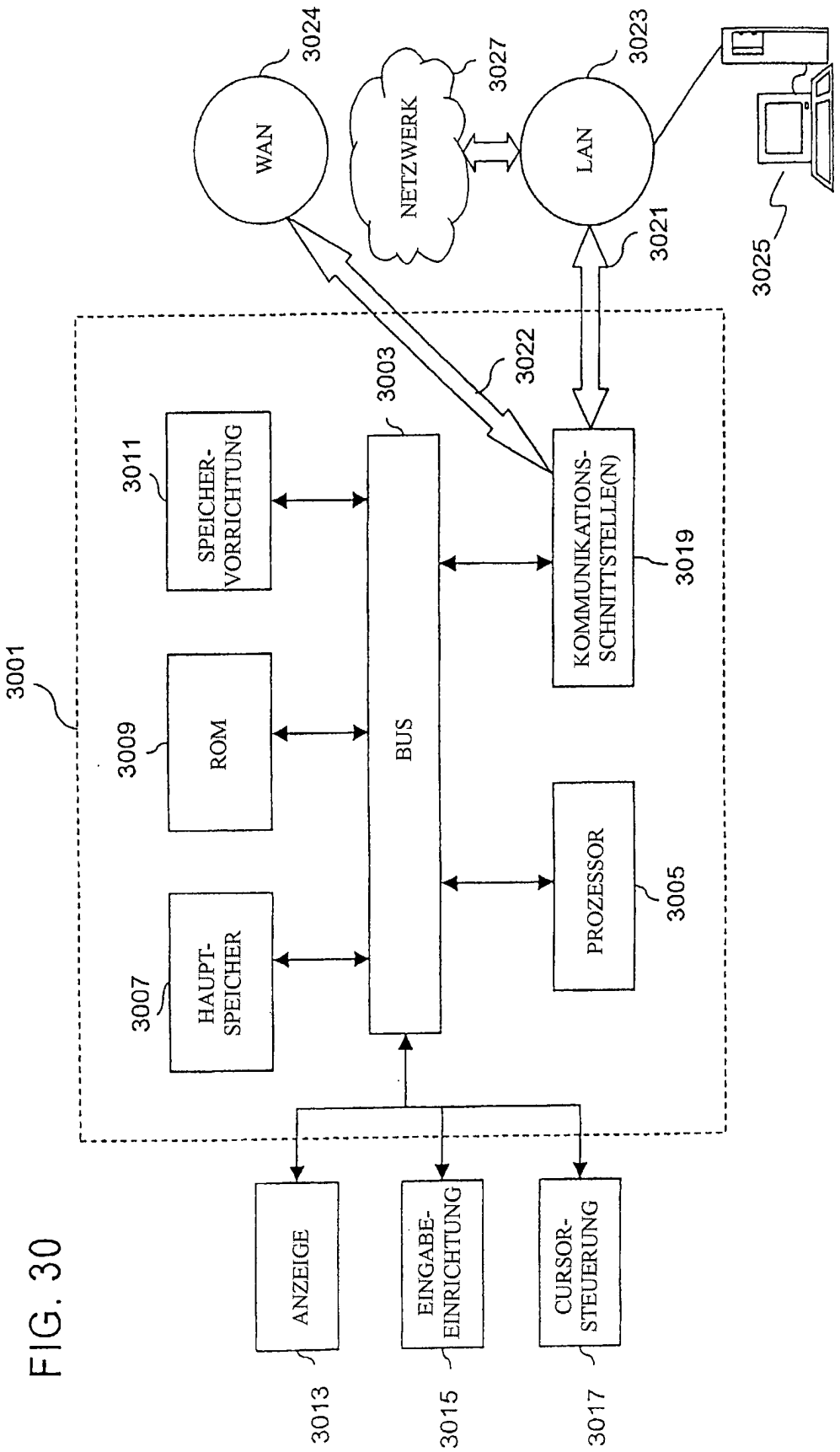

WAN
3024
NETZWERK
3027
LAN
3023
3025
3021
3022
3003
3001
SPEICHER-VORRICHTUNG
3011
ROM
3009
HAUPT-SPEICHER
3007
BUS
KOMMUNIKATIONS-SCHNITTSTELLE(N)
3019
PROZESSOR
3005
ANZEIGE
3013
EINGABE-EINRICHTUNG
3015
CURSOR-STEUERUNG
3017

FIG. 30

FIG. 31

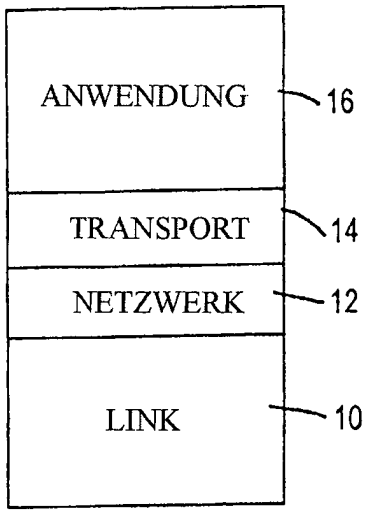
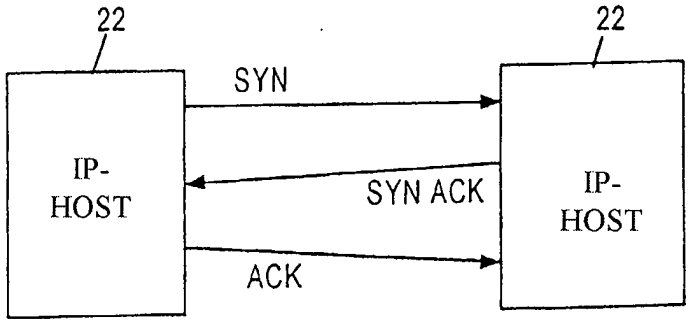


FIG. 32



TCP-VERBINDUNG