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

(10) **DE 601 14 942 T2** 2006.08.17

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 175 045 B1**
(21) Deutsches Aktenzeichen: **601 14 942.4**
(96) Europäisches Aktenzeichen: **01 306 265.8**
(96) Europäischer Anmeldetag: **20.07.2001**
(97) Erstveröffentlichung durch das EPA: **23.01.2002**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **16.11.2005**
(47) Veröffentlichungstag im Patentblatt: **17.08.2006**

(51) Int Cl.[8]: ***H04L 29/06*** (2006.01)
H04L 12/56 (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:
Witte, Weller, Gahlert, Otten & Steil, 70178 Stuttgart

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

(72) Erfinder:
Border, John, Poolesville, Maryland 20837, US;
Gee, Greg, San Diego, California 92122, US

(54) Bezeichnung: **Verfahren und System für das Verwenden eines Kernnetz-Protokolls zur Verbesserung der Netzleistung**

Beschreibung

Gebiet der Erfindung

[0001] Die vorliegende Erfindung ist allgemein auf ein Verfahren und ein System zur Verwendung eines Backbone-Protokolls gerichtet und insbesondere auf das Backbone-Protokoll, das die Netzwerkleistung verbessert.

Hintergrund

[0002] Die Verwurzelung von Datennetzwerken in den Routinen der modernen Gesellschaft, wie durch die Vorherrschaft des Internets bewiesen wird, insbesondere des World Wide Web, hat eine ständig wachsende Anforderung an die Serviceprovider gestellt, ihre Netzwerkleistung kontinuierlich zu verbessern. Um dieser Herausforderung gerecht zu werden, haben Serviceprovider stark in die Aufrüstung ihrer Netzwerke investiert, um die Systemkapazität (d.h. die Bandbreite) zu erhöhen. In vielen Fällen sind solche Upgrades bzw. Aktualisierungen nicht sehr ökonomisch, oder die physikalischen Einschränkungen des Kommunikationssystems erlauben kein einfaches „Aktualisieren". Entsprechend haben Serviceprovider ebenfalls in die Entwicklung von Techniken investiert, um die Leistung ihrer Netzwerke zu optimieren. Da viele der heutigen Netzwerke entweder mit dem Transmission Control Protocol/Internet-Protokoll (TCP/IP) arbeiten, oder mit diesem in Verbindung treten müssen, wurde die Aufmerksamkeit auf die Optimierung der TCP/IP-basierten Netzwerkoperationen fokussiert.

[0003] Als der Netzwerkstandard für das globale Internet hat TCP/IP innerhalb der Industrie eine solche Akzeptanz gewonnen, aufgrund seiner Flexibilität und seinem starken Erbe in der Forschungsgemeinde.

[0004] Das Transmission-Control-Protocol (TCP) ist das dominierende Protokoll, das heutzutage auf dem Internet verwendet wird. TCP wird getragen durch das Internet-Protokoll (IP) und wird in einer Vielzahl von Anwendungen eingesetzt, einschließlich zuverlässiger Dateiübertragungs- und Internet-Webpage-Zugangsanwendungen. Die vier Schichten des TCP/IP-Protokoll-Pakets sind in Fig. 14 dargestellt. Die Verbindungsschicht (oder die Netzwerkschnittstellenschicht) **1410** umfasst, wie dargestellt, Vorrichtungstreiber im Betriebssystem und jeder entsprechenden Netzwerk-Schnittstellenkarte. Zusammen handhaben der Vorrichtungstreiber und die Schnittstellenkarten die Hardware des Teils der physikalischen Verbindung mit irgendeinem Kabel oder einem anderen Medientyp, der verwendet wird. Die Netzwerkschicht (die auch als Internetschicht bezeichnet wird) **1412** handhabt den Transport der Pakete im Netzwerk. Das Routen bzw. Leiten von Paketen findet beispielsweise in der Netzwerkschicht **1412** statt. IP, Internet Control-Message-Protocol (ICMP) und das Internet-Group-Management-Protocol (IGMP) können die Netzwerkschicht in dem PCT/IP-Protokoll-Paket bereitstellen. Die Transportschicht **1414** stellt einen Datenfluss zwischen zwei Hosts für die Anwendungsschicht **1416** zuvor bereit.

[0005] In dem TCP/IP-Protokoll-Paket gibt es zumindest zwei unterschiedliche Transportprotokolle, TCP und ein User-Datagram-Protokoll (UDP). TCP, das einen zuverlässigen Datenfluss zwischen zwei Hosts bereitstellt, betrifft hauptsächlich das Aufteilen der Daten, die von der Anwendungsschicht **1416** zu diesen geleitet werden, in passend bemessene Stücke für die nachfolgend erläuterte Netzwerkschicht **1412**, betrifft das Bestätigen der empfangenen Pakete, das Setzen von Timeouts, um sich zu vergewissern, dass das andere Ende die gesendeten Pakete bestätigt, usw. Da dieser zuverlässige Datenfluss von der Transportschicht **1414** bereitgestellt wird, kann die Anwendungsschicht **1416** diese Details ignorieren. UDP andererseits stellt der Anwendungsschicht **1416** einen viel einfacheren Dienst zur Verfügung. UDP sendet einfach Datenpakete, die Datagramme genannt werden, von einem Host zum anderen, wobei es aber keine Garantie gibt, dass die Datagramme das andere Ende erreichen. Jede gewünschte Zuverlässigkeit muss von der Anwendungsschicht **1416** hinzugefügt werden.

[0006] Die Anwendungsschicht **1416** handhabt die Details der jeweiligen Anwendung. Es gibt viele allgemeine TCP/IP-Anwendungen, die nahezu jede Implementierung bereitstellen. Diese umfassen Telnet für ein Remote-Log-in (Log-in aus der Ferne), 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.

[0007] Wie zuvor beschrieben stellt das TCP eine zuverlässige geordnete Auslieferung von Daten zwischen zwei IP-Hosts bereit. Die IP-Hosts bauen eine TCP-Verbindung auf, indem ein herkömmliches TCP-Drei-Wege-Handshake verwendet wird, und übertragen dann Daten, indem ein fensterbasiertes Protokoll verwendet wird, wobei die erfolgreich empfangenen Daten bestätigt werden.

[0008] Um zu verstehen, wo Optimierungen ausgeführt werden können, ist es aufschlussreich, einen typischen TCP-Verbindungsaufbau zu betrachten.

[0009] Fig. 15 zeigt ein Beispiel des herkömmlichen TCP-Drei-Wege-Handshakes zwischen IP-Hosts **1520** und **1522**. Zuerst sendet der IP-Host **1520**, der eine Übertragung zu dem IP-Host **1522** initiieren möchte, ein Synchronisations(SYN)-Signal an den IP-Host **1522**. Der IP-Host **1522** bestätigt das SYN-Signal vom IP-Host **15020**, indem er eine SYN-Bestätigung (ACK; Acknowledgement) sendet. Der dritte Schritt des herkömmlichen TCP-Drei-Wege-Handshakes ist das Absenden eine ACK-Signals von dem IP-Host **1520** an den IP-Host **1522**. Der IP-Host **1522** ist nun bereit, die Daten vom IP-Host **1520** (und umgekehrt) zu empfangen. Nachdem die gesamten Daten übertragen sind, wird ein anderer Handshake (gleich zu dem Handshake, der zur Initiierung der Verbindung beschrieben wurde) verwendet, um die TCP-Verbindung zu schließen.

[0010] TCP wurde als sehr flexibel entworfen und arbeitet über ein breites Gebiet von Kommunikationsverbindungen einschließlich langsamen und schnellen Verbindungen, Verbindungen mit hoher Latenz, und Verbindungen mit geringen und hohen Fehlerraten. Während TCP (und andere Protokolle hoher Schicht) mit vielen unterschiedlichen Arten von Verbindungen arbeiten, ist jedoch die TCP-Leistung, insbesondere der mögliche Durchsatz über die TCP-Verbindung, durch die Eigenschaften der Verbindung beeinflusst, in der es verwendet wird. Es gibt viele Verbindungsschichtentwurfsbetrachtungen, die berücksichtigt werden sollten, wenn ein Verbindungsschichtdienst entworfen wird, der dazu gedacht ist, Internet-Protokolle zu unterstützen. Allerdings können nicht alle Eigenschaften durch Auswahlen in dem Verbindungsschichtentwurf kompensiert werden. PCP wurde entwickelt, um mit Bezug auf die durchquerten Verbindungen sehr flexibel zu sein. Eine solche Flexibilität wird auf Kosten eines suboptimalen Betriebs in einer Vielzahl von Umgebungen gegenüber einem angepassten Protokoll erreicht. Das angepasste bzw. maßgeschneiderte Protokoll, das gewöhnlicherweise proprietär ist, kann optimaler sein, aber es kann im Hinblick auf Netzwerkumgebungen und Interoperabilität an Flexibilität fehlen.

[0011] Eine Alternative zu einem maßgeschneiderten Protokoll besteht in der Verwendung von leistungsverbessernden Proxies (PEPs), um eine allgemeine Klasse von Funktionen auszuführen, die als „TCP-Spoofing" bezeichnet werden, um die TCP-Leistung über gestörte bzw. beeinträchtigte Verbindungen (d.h. hohe Latenz oder große Fehlerrate) zu verbessern. TCP-Spoofing umfasst eine Zwischennetzwerkvorrichtung (der leistungsverbessernde Proxy (PEP)), die das Verhalten der TCP-Verbindung überwacht und ändert durch das Hinzufügen und/oder Löschen von TCP-Segmenten, um zu versuchen, dessen Leistung zu verbessern.

[0012] Herkömmliche TCP-Spoofing-Implementierungen umfassen das lokale Bestätigen bzw. Acknowledgement von TCP-Datensegmenten, um den TCP-Datensender zu veranlassen, zusätzliche Daten schneller zu senden, als er senden würde, wenn ein Spoofing nicht ausgeführt werden würde, womit der Durchsatz der TCP-Verbindung verbessert wird. Allgemein wurden herkömmliche TCP-Spoofing-Implementierungen einfach auf eine Erhöhung des Durchsatzes der TCP-Verbindungen fokussiert entweder durch Einsatz größerer Fenster über die Verbindung oder durch Einsatz einer Komprimierung, um die Datenmenge zu reduzieren, die gesendet werden muss, oder durch beides.

[0013] Viele TCP/IP-Implementierungen basieren auf einer TCP-ACK-Manipulation. Dies kann umfassen eine TCP-ACK-Beabstandung, bei der ACKs, die zusammengebündelt wurden, getrennt werden, lokale TCP-ACKs, lokale TCP-Neuübertragungen und TCP-ACK-Filterung und Neuaufbau. Andere TCP-Mechanismen umfassen ein Tunnen, Komprimieren, prioritätsbasiertes Multiplexing, ein politikbasiertes Routen und die Fähigkeit, Ausfallsicherungs-Verkehr zu unterstützen.

[0014] Eine Backbone (bzw. Kernnetz)-Verbindung und das zugeordnete Backbone-Protokoll sind Schlüsselelemente einer Netzwerkimplementierung von PEP-Mechanismen. Ein nützliches Backbone-Protokoll sollte:

- eine zuverlässige Datenauslieferung bereitstellen;
- eine relativ kleine Menge an Acknowledgement-Verkehr benutzen; und
- so einfach wie möglich sein, während es dennoch alle Merkmale des Netzwerks unterstützen kann, das es unterstützt;

[0015] Ein weiteres wünschenswertes Merkmal eines Backbone-Protokolls besteht in der Fähigkeit, eine allgemeine Backbone-Verwendung zu unterstützen.

[0016] Einige existierende Protokolle, die als Backbone-Protokolle verwendet werden könnten, umfassen:

- TCP und Varianten;
- das Xpress-Transport-Protokoll (XTP);

- das Message-Multiplexing-Protocol (MEMUX);
- das Satellite-Transport-Protocol (STP);
- das Service-Specific-Connection-Oriented-Protocol (SSCOP);
- das Internet-Reliable-Transport-Protocol (IRTP);
- das Optimum-Data-Link-Protocol (ODLC);
- das Reliable-Data-Protocol (RDP); und
- das Boosted-Seesion-Transport(BST)Protocol.

[0017] Allerdings ist keines der zuvor aufgelisteten Protokolle vollständig für alle Typen von Backbone-Verbindungen geeignet. Beispielsweise ist die TCP-Leistung über Verbindungen begrenzt, die hohe Bandbreitenverzögerungsprodukte besitzen. (Das ist der Grund, warum ein PEP verwendet wird, um ein TCP-Spoofing an erster Stelle auszuführen.) In einigen Fällen spezifiziert auch die Protokollspezifikation nicht vollständig das Protokollverhalten für alle Eigenschaften einer Backbone-Verbindung. Beispielsweise beschreibt die RDP-Spezifikation die Verwendung von NULL-Segmenten, um Verbindungsfehler zu erfassen, gibt aber keine Beschreibung darüber, wie sie zu diesem Zweck verwendet werden sollen.

[0018] Braden et al. beschreibt in „Integrated services in the Internet Architecture: an Overview" (Network working group request for comments, no. 166, Juni 1994) eine Kommunikationsarchitektur mit einem Kommunikationsservicemodell, das durch ein Kommunikationssystem implementiert ist.

[0019] Cisco beschreibt in „Policy-Based Routing" (White Paper, Juli 2000) ein Kommunikationssystem, das Datenpakete auf der Basis von vordefinierten Politiken routet.

[0020] Baras et al. beschreibt in „Fast asymmetric Internet over Wireless Satellite-Terrestrial Networks" (Annual Military Communications Conference, vol. 1. November 1997) ein Satellitenkommunikationssystem, das seine Satellitenverbindung gegenüber den Internethosts durch ein Hybrid-Gateway isoliert.

Zusammenfassung der Erfindung

[0021] Aufgrund der Probleme, die mit der Auswahl eines Backbone (auch Kernnetz genannt)-Protokolls verknüpft sind, unterstützt die vorliegende Erfindung die Verwendung unterschiedlicher Backbone-Protokolle für unterschiedliche Typen von Backbone-Verbindungen. Und die Erfindung unterstützt in einer Konfiguration, bei der mehrere Backbone-Verbindungen unterschiedlichen Typus existieren können, die Verwendung eines unterschiedlichen Backbone-Protokolls für jeden Verbindungstyp zur gleichen Zeit.

[0022] Das Verfahren und das System der vorliegenden Erfindung verwenden ein Backbone-Protokoll, das Verbindungsfehler erfasst, eine dynamische Einstellung der Fenstergröße unterstützt, die Grenze der Anzahl von Neuübertragungen definiert, bevor aufgegeben wird und ein Verbindungsabbruch erklärt wird, und ist gut skalierbar zur Verwendung mit großen Fenstern. Das Backbone-Protokoll der vorliegenden Erfindung erfordert auch keinen langsamen Startalgorithmus, so dass als Ergebnis Bandbreitenbegrenzungen über die Flusssteuerung gehandhabt werden können (über die Pufferplatzverfügbarkeit), die an den Übertragungspunkten ausgeübt wird, und über ein Pufferplatz-Feedback zwischen den Backbone-Protokoll-Partnern.

[0023] Ferner spricht die vorliegende Erfindung das zuvor ausgeführte Bedürfnis an, ein Kommunikationssystem mit einer Leistungsverbesserungsfunktionalität bereitzustellen. Eine Backbone-Verbindungsvorrichtung kommuniziert mit den leistungsverbessernden Proxy (PEP) Endpunktplattformen, um die Plattformen zu konfigurieren, indem Profile verwendet werden, die den PEP-Endplattformen entsprechen.

[0024] Die vorliegende Erfindung ist in den angehängten Ansprüchen definiert.

[0025] Ein Verfahren zum Leiten bzw. Routen von Information in einem Kommunikationssystem, das eine Plattform und eine Backbone-Verbindungsvorrichtung umfasst, die konfiguriert ist, um eine Vielzahl von leistungsverbessernden Funktionen auszuführen, ist vorgesehen. Das Verfahren umfasst ein Empfangen der Information von der Plattform und ein Empfangen von Backbone-Verbindungsparametern, wobei die Backbone-Verbindungsvorrichtung ein Profil aufrecht erhält, das die Backbone-Verbindungsparameter enthält und die Information entsprechend dem Profil routet.

[0026] Eine Plattform, die konfiguriert ist, um leistungsverbessernde Funktionen bereitzustellen, ist vorgesehen. Die Plattform liefert Information und Backbone-Verbindungsparameter an eine Backbone-Verbindungsvorrichtung. Die Backbone-Verbindungsvorrichtung umfasst ein Profil, das die Backbone-Verbindungsparame-

ter spezifiziert, wobei das Kommunikationssystem konfiguriert ist, um die Information entsprechend einem Profil zu routen.

[0027] Eine Backbone-Verbindungsvorrichtung zum Routen von Information in einem Kommunikationssystem ist vorgesehen, das eine Plattform aufweist, die konfiguriert ist, um eine Vielzahl von leistungsverbessernden Funktionen auszuführen. Die Backbone-Verbindungsvorrichtung umfasst Mittel zum Empfangen von Information und Backbone-Verbindungsparametern, Mittel zum Aufrechterhalten eines Profils, das die Backbone-Verbindungsparameter enthält, und Mittel zum Routen der Information entsprechend dem Profil.

[0028] Ein computerlesbares Medium, das ein oder mehrere Sequenzen eines oder mehrerer Befehle zum Routen von Information in einem Kommunikationssystem trägt, das eine Plattform umfasst, die konfiguriert ist, um eine Vielzahl von leistungsverbessernden Funktionen auszuführen, ist vorgesehen. Das computerlesbare Medium trägt eine oder mehrere Sequenzen eines oder mehrerer Befehle, die, wenn sie von einem oder mehreren Prozessoren ausgeführt werden, den einen oder mehrere Prozessoren veranlassen, die Schritte auszuführen: Empfangen der Information von der Plattform und Empfangen von Backbone-Verbindungsparametern, wobei die Backbone-Verbindungsvorrichtung ein Profil aufrecht erhält, das die Backbone-Verbindungsparameter enthält, und Routen der Information entsprechend dem Profil.

Kurze Beschreibung der Zeichnungen

[0029] Eine vollständige Würdigung der Erfindung und viele begleitende Vorteile davon ergeben sich leicht aus der nachfolgenden detaillierten Beschreibung, wenn sie in Verbindung mit den begleitenden Zeichnungen betracht wird, wobei:

[0030] Fig. 1 ein Diagramm eines Kommunikationssystems ist, in dem der leistungsverbessernde Proxy (PEP) der vorliegenden Erfindung implementiert ist;

[0031] Fig. 2 ein Diagramm einer PEP-Endpunkt-Plattformumgebung ist entsprechend einer Ausführungsform der vorliegenden Erfindung;

[0032] Fig. 3 ein Diagramm eines PCT-Spoofing-Kernels (TSK) ist, der in der Umgebung von Fig. 2 verwendet wird;

[0033] Fig. 4A und Fig. 4B Flussdiagramme des Verbindungsaufbaus mit einem Drei-Wege-Handshake-Spoofing bzw. ohne ein Drei-Wege-Handshake-Spoofing sind;

[0034] Fig. 5 ein Diagramm eines PEP-Paketflusses zwischen zwei PEP-Endpunkten entsprechend einer Ausführungsform der vorliegenden Erfindung ist;

[0035] Fig. 6 ein Diagramm eines IP (Internet-Protokoll) Paketflusses durch einen PEP-Endpunkt entsprechend einer Ausführungsform der vorliegenden Erfindung ist;

[0036] Fig. 7 ein Diagramm von PEP-Endpunkt-Profilen ist, die in der Plattform von Fig. 2 verwendet werden;

[0037] Fig. 8 ein Diagramm der Schnittstellen eines PEP-Endpunkts ist, der als ein IP-Gateway entsprechend einer Ausführungsform der vorliegenden Erfindung implementiert ist;

[0038] Fig. 9 ein Diagramm der Schnittstellen eines PEP-Endpunkts ist, der als Multimedia-Relay implementiert ist, entsprechend einer Ausführungsform der vorliegenden Erfindung;

[0039] Fig. 10 ein Diagramm der Schnittstellen eines PEP-Endpunkts ist, der als Multimedia-VSAT (Very-Small-Aperture-Terminal) implementiert ist, entsprechend einer Ausführungsform der vorliegenden Erfindung;

[0040] Fig. 11 ein Diagramm der Schnittstellen eines PEP-Endpunkts ist, der in einer Bodenstation implementiert ist, entsprechend einer Ausführungsform der vorliegenden Erfindung;

[0041] Fig. 12 ein Diagramm einer BCB-Abbildungs- bzw. Mappingtabelle ist;

[0042] Fig. 13 ein Diagramm eines Computersystems ist, das die PEP-Funktionen ausführen kann, entspre-

chend einer Ausführungsform der vorliegenden Erfindung;

[0043] Fig. 14 ein Diagramm der Protokollschichten des TCP/IP-Protokollpakets ist; und

[0044] Fig. 15 ein Diagramm eines herkömmlichen TCP-Drei-Wege-Handshakes zwischen den IP-Hosts ist.

Detaillierte Beschreibung der bevorzugten Ausführungsformen

[0045] In der nachfolgenden Beschreibung sind zu Erläuterungszwecken spezifische Details ausgeführt, um ein tieferes Verständnis der Erfindung zu liefern. Es ergibt sich jedoch, dass die Erfindung ohne diese spezifischen Details ausgeführt werden kann. In einigen Beispielen sind gut bekannte Strukturen und Vorrichtungen in Blockdiagrammform dargestellt, um ein unnötiges Verschleiern der Erfindung zu vermeiden.

[0046] Obgleich die vorliegende Erfindung mit Bezug auf das Internet und das TCP/IP-Protokollpaket diskutiert wird, ist die vorliegende Erfindung auf andere paketvermittelte Netzwerke und äquivalente Protokolle anwendbar.

[0047] Fig. 1 zeigt ein beispielhaftes Netzwerk **100**, in welchem der leistungsverbessernde Proxy (PEP) der vorliegenden Erfindung verwendet werden kann. Das Netzwerk **100** in Fig. 1 umfasst einen oder mehrere Hosts **110**, die mit einem Netzwerk-Gateway **120** über TCP-Verbindungen verbunden sind. Das Netzwerk-Gateway **120** ist mit einem anderen Netzwerk-Gateway **140** verbunden über eine Backbone-Verbindung auf einem Backbone-Link **130**. Wie in Fig. 1 zu sehen ist, ist der Backbone-Link **130** in einer beispielhaften Ausführungsform als Satelliten-Link bzw. Verbindung gezeigt, die über einen Satelliten **101** aufgebaut wird; ein Fachmann erkennt jedoch, dass andere Netzwerkverbindungen implementiert werden können. Beispielsweise können diese Netzwerkverbindungen über ein allgemeines drahtloses Kommunikationssystem aufgebaut werden (beispielsweise Funknetzwerke, Mobilfunknetzwerke, etc.) oder ein terrestrisches Kommunikationssystem. Das Netzwerk-Gateway **140** ist ferner mit einer zweiten Gruppe von Hosts **150** ebenfalls über TCP-Verbindungen verbunden. In der in Fig. 1 dargestellten Anordnung erleichtern die Netzwerk-Gateways **120**, **140** die Kommunikation zwischen den Gruppen von Hosts **110**, **150**.

[0048] Die Netzwerk-Gateways **120**, **140** erleichtern die Kommunikation zwischen den zwei Gruppen von Hosts **110**, **150**, indem eine Anzahl von leistungsverbessernden Funktionen ausgeführt wird. Diese Netzwerk-Gateways **120**, **140** können ein selektives TCP-Spoofing ausführen, das eine flexible Konfiguration der jeweiligen TCP-Verbindungen ermöglicht, die zu spoofen sind. Zusätzlich verwenden Gateways **120**, **140** ein TCP-Drei-Wege-Handshake, bei dem die TCP-Verbindungen an jedem Ende des Backbone-Links **130** terminiert werden. Lokale Datenbestätigungen bzw. Acknowledgements werden von den Netzwerk-Gateways **120**, **140** verwendet, um damit die TCP-Fenster auf lokale Geschwindigkeit zu erhöhen.

[0049] Die Netzwerk-Gateways **120**, **140** multiplexen ferner mehrere TCP-Verbindungen über eine einzelne Backbone-Verbindung; diese Fähigkeit reduziert die Menge an Acknowledgement-Verkehr, der mit den Daten von mehreren TCP-Verbindungen verknüpft ist, da eine einzelne Backbone-Verbindungsbestätigung verwendet werden kann. Die Multiplexfunktion stellt ebenfalls eine Unterstützung für TCP-Verbindungen mit hohem Durchsatz bereit, wobei das Backbone-Verbindungsprotokoll für den einzelnen verwendeten Backbone-Link optimiert wird. Die Netzwerk-Gateways **120**, **140** unterstützen ebenfalls eine Datenkomprimierung über den Backbone-Link **130**, um die Menge von zu sendendem Verkehr zu reduzieren, um ferner die Möglichkeiten der Backbone-Verbindung weiter auszureizen. Die Netzwerk-Gateways **120**, **140** benutzen ferner eine Datenverschlüsselung in der Datenübertragung über den Backbone-Link **130**, um die Privatsphäre der Daten zu schützen und einen priorisierten Zugang auf die Backbone-Link-Kapazität auf einer TCP-Verbindungsbasis bereitzustellen. Jedes der Netzwerk-Gateways **120**, **140** kann einen bestimmten Pfad für die Daten auswählen, die mit einer Verbindung verknüpft sind, über die die Daten fließen sollen. Die vorherige Fähigkeit der Netzwerk-Gateways **120**, **140** wird nachfolgend vollständiger beschrieben.

[0050] Fig. 2 zeigt einen leistungsverbessernden Proxy (PEP) **200**, der in einem Netzwerk-Gateway **120**, **140** entsprechend einer Ausführungsform der vorliegenden Erfindung implementiert ist. In dieser Ausführungsform besitzt der PEP **200** eine Plattformumgebung **210**, die die Hardware und das Software-Betriebssystem umfasst. Der PEP **200** umfasst ebenfalls Schnittstellen zum Local-Area-Network (LAN) **220**, und Wide-Area-Network(WAN)Schnittstellen **230**. In dem in Fig. 1 gezeigten Beispiel kann das Netzwerk-Gateway **120** die TCP-Verbindungen mit den IP-Hosts **110** über eine lokale LAN-Schnittstelle **220** aufbauen, und kann eine Backbone-Verbindung mit dem Netzwerk-Gateway **140** über eine WAN-Schnittstelle **230** aufbauen. Die PEP-Plattformumgebung **210** kann ebenfalls allgemeine funktionale Module umfassen: ein Routingmodul **240**,

ein Bufferverwaltungsmodul **250**, ein Eventverwaltungsmodul **260**, und ein Parameterverwaltungsmodul **270**. Wie in Fig. 2 gezeigt, umfasst das Netzwerk-Gateway ebenfalls einen TCP-Spoofingkernel bzw. -Kern (TSK) **280**, einen Backbone-Protokollkernel (BPK) **282**, einen Priorisierungskernel (PK) 284 und einen Pfadauswahl-Kernel (PSK) **286**. Diese vier Kernels bilden im Wesentlichen die Funktionalität des leistungsverbessernden Proxy **200**.

[0051] Die Plattformumgebung **210** umfasst eine Anzahl von Funktionen. Eine solche Funktion besteht darin, die verschiedenen PEP-Kernels **280**, **282**, **284**, **286** gegenüber spezifischen Implementierungsbeschränkungen abzuschirmen. Das heißt, dass die Plattformumgebung **210** Funktionen ausführt, die die verschiedenen PEP-Kernels **280**, **282**, **284**, **286** nicht direkt ausführen können, da die Implementierung der Funktion plattformspezifisch ist. Die Anordnung hat den vorteilhaften Effekt, dass die plattformspezifischen Details gegenüber den PEP-Kernels **280**, **282**, **284**, **286** versteckt werden, was die PEP-Kernels portierbarer macht. Ein Beispiel einer plattformspezifischen Funktion ist die Belegung eines Puffers. Bei einigen Plattformen werden die Puffer erzeugt, wenn sie benötigt werden, während bei anderen Plattformen die Puffer beim Start erzeugt werden und in verlinkten Listen für eine spätere Verwendung organisiert werden. Es ist anzumerken, dass die plattformspezifischen Funktionen nicht auf die Funktionen beschränkt sind, die allen Kernels **280**, **282**, **284**, **286** gemein sind. Eine Funktion, die für einen bestimmten Kernel spezifisch ist, beispielsweise die Belegung eines Steuerungsblocks bzw. Kontrollblocks für das TCP-Spoofing, kann ebenfalls in der Plattformumgebung implementiert sein, um die plattformspezifischen Details gegenüber dem Kernel zu verstecken.

[0052] In einer beispielhaften Ausführungsform stellt die Plattformumgebung **210** das Aufgaben- bzw. Task-Umfeld bereit, in dem die PEP-Kernel **280**, **282**, **284**, **286** laufen. In einer anderen beispielhaften Ausführungsform können alle PEP-Kernels **280**, **282**, **284**, **286** in dem gleichen Task-Umfeld aus Effizienzgründen laufen, dies ist jedoch nicht notwendig.

[0053] Ferner stellt die Plattformumgebung **210** in einer beispielhaften Ausführungsform eine Schnittstelle zwischen der PEP-Funktionalität (in den Kernels **280**, **282**, **284**, **286** verkörpert) und der anderen Funktionalität des Netzwerk-Gateways **120**, **140** bereit. Die Plattformumgebung **210** kann die Schnittstelle zwischen der PEP-Funktionalität und der Routingfunktion **240** bereitstellen, wie in Fig. 2 zu sehen ist. Es ist anzumerken, dass die plattformspezifischen Funktionen, die in Fig. 2 gezeigt sind, Beispiele sind und nicht als abschließende Liste zu betrachten sind. Es ist ferner anzumerken, dass die PEP-Kernel, die in Fig. 2 als sich einander berührend dargestellt sind (**280**, **282** und **284**, **286**), eine direkte Prozedur-Schnittstelle zueinander haben können. Ferner können die Kernel **280**, **282**, **284**, **286** direkte Schnittstellen aufweisen, um die Leistung zu verbessern, im Gegensatz zu einem Weiterleiten jeglicher Daten durch die Plattformumgebung **210** (wie in Fig. 2 gezeigt).

[0054] Zusätzlich zu den PEP-Kernels **280**, **282**, **284** und **286** kann die PEP-Endpunktplattform **210** einen Datenkomprimierungs-Kernel (CK) **290** und einen Verschlüsselungs-Kernel (EK) **282** aufweisen. Diese Kernels **280**, **282**, **284**, **286**, **290** und **292**, wie zuvor beschrieben, leisten die Kommunikation zwischen den zwei Gruppen von Hosts **110**, **150**, indem eine Vielzahl von leistungsverbessernden Funktionen ausgeführt werden, entweder einzeln oder in Kombination. Diese leistungsverbessernden Funktionen umfassen ein selektives TCP-Spoofing, Drei-Wege-Handshake-Spoofing, lokales Daten-Acknowledgement, TCP-Verbindungs- zu Backbone-Verbindungs-Multiplexen, Datenkomprimierung/Verschlüsselung, Priorisierung und Pfadauswahl.

[0055] Ein selektives TCP-Spoofing wird von dem TSK **280** ausgeführt und umfasst eine Menge von benutzerkonfigurierbaren Regeln, die eingesetzt werden, um festzulegen, welche TCP-Verbindungen gespooft werden sollten. Ein selektives TCP-Spoofing verbessert die Leistung, indem TCP-Spoofing bezogene Quellen, wie beispielsweise Pufferplatz, Controlblöcke etc. nicht für TCP-Verbindungen herangezogen werden, für die der Benutzer festgelegt hat, dass ein Spoofing nicht vorteilhaft ist oder erforderlich ist, und indem die Verwendung maßgeschneiderter Parameter für TCP-Verbindungen unterstützt wird, die gespooft werden.

[0056] Insbesondere unterscheidet der TSK **280** unter den verschiedenen TCP-Verbindungen, basierend auf den ihn verwendenden Anwendungen. Das heißt, der TSK **280** unterscheidet unter den TCP-Verbindungen, um festzulegen, welche Verbindung gespooft werden soll sowie die Art und Weise, in der die Verbindung gespooft werden soll; beispielsweise ob der Drei-Wege-Handshake gespooft werden soll, den bestimmten Timeout-Parameter für die gespoofte Verbindung, etc. TCP-Spoofing wird dann nur für jene TCP-Verbindungen ausgeführt, die mit Anwendungen verknüpft sind, für die ein hoher Durchsatz oder reduzierte Verbindungsstartlatenz (oder beides) erforderlich sind. Als Ergebnis spart der TSK **280** TCP-Spoofingquellen nur für jene TCP-Verbindungen ein, für die ein hoher Durchsatz oder eine reduzierte Verbindungsstartlatenz (oder beides) gefordert wird. Ferner erhöht der TSK **280** die Gesamtzahl der TCP-Verbindungen, die aktiv sein können, be-

vor die TCP-Spoofingquellen ausgehen, da jede aktive TCP-Verbindung, die keinen hohen Durchsatz erfordert, keine belegten Ressourcen sind.

[0057] Ein Kriterium zum Identifizieren von TCP-Verbindungen von Anwendungen, für die TCP-Spoofing ausgeführt und nicht ausgeführt werden sollte, ist das TCP-Portnummernfeld, das in den gesendeten TCP-Paketen enthalten ist. Allgemein werden eindeutige Portnummern jedem Anwendungstyp zugeordnet. Welche TCP-Portnummern gespooft und nicht gespooft werden sollen, kann in dem TSK **280** abgespeichert werden. Der TSK **280** wird ebenfalls neu konfiguriert, um es einem Benutzer oder Operator zu ermöglichen, die TCP-Portnummern neu zu konfigurieren, die gespooft oder nicht gespooft werden sollen. Der TSK **280** erlaubt auch einem Benutzer oder Operator, zu steuern, welche TCP-Verbindungen gespooft werden sollen, basierend auf anderen Kriterien. Im Allgemeinen kann eine Entscheidung, ob eine TCP-Verbindung gespooft wird, auf irgendeinem Feld innerhalb eines TCP-Pakets basieren. Der TSK **280** erlaubt es einem Benutzer, zu spezifizieren, welches Feld geprüft werden soll und welcher Wert in diesen Feldern TCP-Verbindungen identifiziert, die gespooft oder nicht gespooft werden sollen. Ein anderes Beispiel einer möglichen Benutzung dieser Fähigkeit besteht darin, dass der Benutzer oder Operator die IP-Adresse des TCP-Pakets auswählt, um zu steuern, für welche Benutzer TCP-Spoofing ausgeführt wird. Der TSK **280** ermöglicht es auch einem Benutzer, mehrere Felder zur gleichen Zeit zu betrachten. Als Ergebnis erlaubt der TSK **280** einem Benutzer oder Operator, mehrere Kriterien zum Auswählen der zu spoofenden TCP-Verbindungen einzusetzen. Beispielsweise kann der Systemoperator durch Auswahl sowohl der IP-Adresse als auch der TCP-Portnummernfelder ein TCP-Spoofing nur durch spezifische Anwendungen von spezifischen Benutzern ermöglichen.

[0058] Die benutzerkonfigurierbaren Regeln können fünf beispielhafte Kriterien umfassen, die von dem Benutzer oder Operator spezifiziert werden beim Erzeugen einer selektiven TCP-Spoofingregel: Ziel-IP-Adresse; Quellen-IP-Adresse; TCP-Portnummern (die sowohl auf das TCP-Ziel als auch Quellportnummern anwendbar sind); TCP-Optionen, und das IP-differentiated-Services (DS); unterschiedliche Dienstefeld. Wie zuvor ausgeführt können jedoch andere Felder innerhalb des TCP-Pakets verwendet werden.

[0059] Wie zuvor diskutiert können zusätzlich zu der Unterstützung selektiver TCP-Spoofingregeln für jedes dieser Kriterien UND- oder ODER-Kombinationsoperatoren verwendet werden, um die Kriterien miteinander zu verknüpfen. Beispielsweise kann durch Verwendung des UND-Kombinationsoperators eine Regel definiert werden, um ein TCP-Spoofing für FTP-Daten zu sperren, die von einem spezifischen Host empfangen werden. Ebenfalls kann die Reihenfolge, in der die Regeln spezifiziert werden, wesentlich sein. Es ist für eine Verbindung möglich, die Kriterien mehrerer Regeln zu erfüllen. Deshalb können vom TSK **280** Regeln in der vom Operator spezifizierten Reihenfolge angewendet werden, wobei die Aktion der ersten passenden Regel verwendet wird. Eine Default- bzw. Standardregel kann ebenfalls gesetzt werden, die die Aktion definiert, die für TCP-Verbindungen auszuführen ist, die auf keine der definierten Regeln passen. Der Satz von Regeln, der von dem Operator ausgewählt wird, kann in einem selektiven TCP-Spoofing-Auswahlprofil definiert werden.

[0060] Als Beispiel sei angenommen, dass ausreichend Pufferplatz belegt wurde, um fünf TCP-Verbindungen zu spoofen, falls vier Anwendungen mit niedriger Geschwindigkeit (d.h. Anwendungen, die aufgrund ihrer Natur keine Hochgeschwindigkeit erfordern) Verbindungen aufbauen, zusammen mit einer Hochgeschwindigkeitsanwendung, wobei die Hochgeschwindigkeitsverbindung Zugriff auf etwa 1/5 des verfügbaren Spoofing-Pufferplatzes hat. Falls fünf Verbindungen mit niedriger Geschwindigkeit vor der Hochgeschwindigkeitsverbindung aufgebaut werden, kann die Hochgeschwindigkeitsverbindung überhaupt nicht gespooft werden. Indem der TSK **280** Auswahl-Spoofing-Mechanismus verwendet wird, belegen die Verbindungen mit niedriger Geschwindigkeit keinen Spoofing-Pufferplatz. Deshalb hat die Hochgeschwindigkeitsverbindung immer Zugriff auf den gesamten Pufferplatz, so dass dessen Leistung mit Bezug auf eine Implementierung ohne das selektive TCP-Spoofing-Merkmal des TSK **280** verbessert wird.

[0061] Der TSK **280** erleichtert ebenfalls das Spoofen des herkömmlichen Drei-Wege-Handshakes. Ein Drei-Wege-Handshake-Spoofing umfasst das lokale Antworten auf eine Verbindungsanfrage, um eine TCP-Verbindung aufzubauen parallel mit dem Vorwärtsleiten der Verbindungsanforderungen über den Backbone-Link **130** (Fig. 1). Dies ermöglicht es dem entsprechenden IP-Host (beispielsweise **110**) den Punkt zu erreichen, an dem er in der Lage ist, die Daten zu senden, die mit lokalen Geschwindigkeiten gesendet werden müssen, das heißt Geschwindigkeiten, die unabhängig sind von der Latenz des Backbone-Links **130**. Ein Drei-Wege-Handshake-Spoofing erlaubt es, dass die Daten, die der IP-Host **110** senden soll, an den Ziel-IP-Host **150** gesendet wird, ohne auf den Endaufbau der TCP-Verbindung zu warten. Für Backbone-Links **130** mit großer Latenz reduziert sich signifikant die Zeit, die benötigt wird, um die TCP-Verbindung aufzubauen und noch wichtiger, die Gesamtzeit, die benötigt wird, um eine Antwort (von einem IP-Host **150**) an den IP-Host **110** zu senden.

[0062] Ein spezifisches Beispiel, bei dem diese Technik nützlich ist, betrifft eine Internet-Webpage-Zugangsanwendung. Bei einem Drei-Wege-Handshake-Spoofing kann eine Anforderung von einem IP-Host eine Webpage zu suchen, auf dem Weg zu einem Webserver sein, ohne auf den Aufbau einer End-zu-End-TCP-Verbindung zu warten, so dass die Zeit reduziert wird, die benötigt wird, um die Webpage herunterzuladen.

[0063] Bei einer lokalen Datenbestätigung bestätigt der TSK **280** in dem Netzwerk-Gateway **120** lokal Datensegmente, die von dem IP-Host **120** empfangen werden. Dies ermöglicht, dass der sendende IP-Host **110** zusätzliche Daten sofort sendet. Wichtiger ist noch, dass TCP empfangene Bestätigungen als Signale zur Erhöhung der aktuellen TCP-Fenstergröße benutzt. Als Ergebnis ermöglicht das lokale Senden von Bestätigungen, dass der sendende IP-Host **110** sein TCP-Fenster mit einer viel größeren Geschwindigkeit erhöht, als dies durch End-zu-End-TCP-Bestätigungen getragen würde. Der TSK **280** (der Spoofer) übernimmt die Verantwortung für eine sichere Verteilung bzw. Übermittlung von Daten, die er bestätigt hat.

[0064] In dem BPK **282** werden mehrere TCP-Verbindungen auf eine einzelne Backbone-Verbindung gemultiplext und von dieser getragen. Dies verbessert die Systemleistung, indem die Daten für mehrere TCP-Verbindungen durch eine einzelne Backbone-Verbindungs-Bestätigung (ACK) bestätigt werden, was die Menge des Bestätigungsverkehrs signifikant reduziert, der erforderlich ist, um einen hohen Durchsatz über den Backbone-Link **130** aufrecht zu erhalten. Zusätzlich wählt der BPK **282** ein Backbone-Verbindungsprotokoll aus, das optimiert wird, um einen hohen Durchsatz für die bestimmte Verbindung bereitzustellen. Verschiedene Backbone-Verbindungsprotokolle können von dem BPK **282** bei unterschiedlichen Backbone-Links verwendet werden, ohne die grundlegende TCP-Spoofing-Implementierung zu verändern. Das Backbone-Verbindungs-Protokoll, das von dem BPK **282** ausgewählt wird, stellt eine geeignete Unterstützung für eine zuverlässige Hochgeschwindigkeitsübertragung von Daten über den Backbone-Link **130** bereit, versteckt die Details der Verschlechterung (beispielsweise hohe Latenz) des Links bzw. Verbindung gegenüber der TCP-Spoofing-Implementierung.

[0065] Das Multiplexen durch den BPK **282** ermöglicht die Verwendung eines Backbone-Link-Protokolls, das individuell maßgeschneidert ist für die Verwendung mit einem bestimmten Link und stellt eine Technik dar, um die Leistung des Backbone-Linkprotokolls mit sehr viel geringerer Abhängigkeit von der individuellen Leistung der TCP-Verbindungen auszureizen, die gespooft werden sollen, als bei herkömmlichen Verfahren. Ferner macht die Fähigkeit, das Backbone-Protokoll für unterschiedliche Backbone-Links anpassen zu können, die vorliegende Erfindung anwendbar für viele unterschiedliche Systeme.

[0066] Der PEP **200** kann optional einen Datenkomprimierungs-Kernel **290** zur Komprimierung von TCP-Daten umfassen und einen Verschlüsselungs-Kernel **282** zum Verschlüsseln der TCP-Daten. Die Datenkomprimierung erhöht die Datenmenge, die über die Backbone-Verbindung getragen werden kann. Unterschiedliche Komprimierungsalgorithmen können von dem Datenkomprimierungs-Kernel **290** unterstützt werden, und mehr als ein Komprimierungstyp kann gleichzeitig unterstützt werden. Der Datenkomprimierungs-Kernel **290** kann optional eine Komprimierung auf der Basis einer TCP-Verbindung anwenden, bevor die TCP-Daten mehrerer TCP-Verbindungen auf die Backbone-Verbindung gemultiplext werden, oder auf der Basis einer Backbone-Verbindung, nachdem die TCP-Daten mehrerer TCP-Verbindungen auf die Backbone-Verbindung gemultiplext wurden. Welche Option verwendet wird, wird dynamisch basierend auf benutzerkonfigurierten Regeln und dem spezifischen Komprimierungsalgorithmus, der eingefügt wird, bestimmt. Beispielhafte Datenkomprimierungsalgorithmen sind offenbart in US Patent Nr. 5,973,630, 5,955,976. Der Verschlüsselungs-Kernel **282** verschlüsselt die TCP-Daten für eine sichere Übertragung über den Backbone-Link **130**. Eine Verschlüsselung kann mit jeder herkömmlichen Technik ausgeführt werden. Es versteht sich auch, dass der zugehörige Spoofer (in dem zuvor ausgeführten Beispiel das Netzwerk-Gateway **140**) passende Kernels zur Dekomprimierung und Entschlüsselung umfasst, die beide mit herkömmlichen Techniken ausgeführt werden können.

[0067] Der PK **284** stellt einen priorisierten Zugang auf die Backbone-Link-Kapazität bereit. Beispielsweise kann die Backbone-Verbindung in N (N > 1) unterschiedliche Teilverbindungen aufgetrennt werden, deren jede einen unterschiedlichen Prioritätswert besitzt. Bei einer beispielhaften Ausführungsform können vier Prioritätswerte unterstützt werden. Der PK **284** benutzt benutzerdefinierte Regeln, um unterschiedliche Prioritäten zuzuordnen, und damit unterschiedliche Teilverbindungen der Backbone-Verbindung an unterschiedliche TCP-Verbindungen. Es ist anzumerken, dass der PK **284** ebenfalls nicht TCP-Verkehr priorisieren kann (beispielsweise UDP (User-Datagram-Protocol) Verkehr), bevor der Verkehr über den Backbone-Link **130** gesendet wird.

[0068] Der PK **284** benutzt ebenfalls benutzerdefinierte Regeln, um zu steuern, wie viel Kapazität der Back-

bone-Link **130** für jeden Prioritätswert verfügbar hat. Beispielhafte Kriterien, die verwendet werden können, um die Priorität zu bestimmen, umfassen die folgenden: Ziel-IP-Adresse; Quell-IP-Adresse; IP-Next-Protocol; TCP-Portnummern (die sowohl auf die TCP-Ziel- als auch die Quell-Portnummer anwendbar sind); UDP-Portnummern (die sowohl für die UDP-Ziel- als auch Quell-Portnummern anwendbar sind); und das IP-differentiated-Services(DS)Feld. Der Datentyp in den TCP-Datenpaketen kann ebenfalls als Kriterium verwendet werden. Beispielsweise könnte diesen Daten höchste Priorität gegeben werden. Übertragungskritische Daten könnten ebenfalls höchste Priorität erhalten. Wie bei einem selektiven TCP-Spoofing kann jedes Feld in dem IP-Paket von PK **284** verwendet werden, um die Priorität zu bestimmen. Allerdings ist anzumerken, dass bei bestimmten Szenarien die Konsequenz der Verwendung eines solchen Feldes darin bestehen kann, dass unterschiedliche IP-Pakete des gleichen Flusses (beispielsweise TCP-Verbindung) unterschiedliche Prioritäten zugeordnet bekommen; diese Szenarien sollten vermieden werden.

[0069] Wie zuvor erwähnt, können zusätzlich zur Unterstützung selektiver Priorisierungsregeln für jedes dieser Kriterien UND- und ODER-Kombinationsoperatoren verwendet werden, um die Kriterien miteinander zu verbinden. Beispielsweise kann die Verwendung des UND-Kombinationsoperators eine Regel definieren, um eine Priorität für SNMP-Daten zuzuweisen, die von einem bestimmten Host empfangen werden. Ebenfalls kann die Reihenfolge, in der die Regeln spezifiziert werden, wesentlich sein. Es ist für eine Verbindung möglich, Kriterien mehrerer Regeln zu erfüllen. Deshalb kann der PK **284** Regeln in der Reihenfolge anwenden, die von dem Operator spezifiziert sind, wobei die Aktion der ersten Regel genommen wird, die passt. Eine Standard- bzw. Default-Regel kann ebenfalls eingestellt werden, die die Aktion definiert, die für IP-Pakete zu verwenden ist, die keine der definierten Regeln erfüllen. Der Satz von Regeln, der von dem Operator ausgewählt wird, kann in einem Priorisierungsprofil definiert werden.

[0070] Im Hinblick auf die Pfadauswahl-Funktionalität ist der PSK **286** verantwortlich für die Bestimmung, welcher Pfad ein IP-Paket nehmen soll, um sein Ziel zu erreichen. Der von dem PSK **286** ausgewählte Pfad kann festgelegt werden, indem die Pfadauswahlregeln angewendet werden. Der PSK **286** bestimmt ebenfalls, welche IP-Pakete weitergeleitet werden sollen, indem ein alternativer Pfad verwendet wird, und welche IP-Pakete fallengelassen werden sollen, wenn einer oder mehrere Hauptpfade ausfallen. Pfadausfallparameter können ebenfalls konfiguriert werden, indem Profile verwendet werden. Die Pfadauswahlregeln können gestaltet werden, um Flexibilität mit Bezug auf die Zuweisung von Pfaden bereitzustellen, während sichergestellt wird, dass alle Pakete, die den gleichen Verkehrsstrom betreffen (beispielsweise die gleiche TCP-Verbindung), den gleichen Pfad nehmen (obgleich es möglich ist, Segmente der gleichen TCP-Verbindung über unterschiedliche Pfade zu senden, wobei dieses Segment-"Auftrennen" negative Nebenwirkungen haben kann). Beispielhafte Kriterien, die verwendet werden können, um einen Pfad auszuwählen, umfassen die folgenden: Priorität des IP-Pakets, wie von der PK **284** eingestellt (sollte das üblichste Kriterium sein); Ziel-IP-Adresse; Quell-IP-Adresse; IP Next Protocol; TCP-Portnummern (was sowohl auf die TCP-Ziel- als auch Quell-Portnummern angewendet werden kann); UDP-Portnummern (was sowohl auf die UDP-Ziel- als auch Quellportnummern angewendet werden kann); und das IP Differentiated Services (DS) Feld. Ähnlich zu dem selektiven TCP-Spoofing und unter Priorisierung kann der PSK **284** einen Pfad bestimmen, in dem ein beliebiges Feld in dem IP-Paket verwendet wird.

[0071] Wie bei den Priorisierungskriterien (Regeln) können die UND- und ODER-Kombinationsoperatoren verwendet werden, um Kriterien miteinander zu verknüpfen. Beispielsweise kann unter Verwendung des UND-Kombinationsoperators eine Regel definiert werden, um einen Pfad für SNMP-Daten auszuwählen, die von einem spezifischen Host empfangen werden. Ebenfalls ist die Reihenfolge, in der die Regeln spezifiziert werden, wesentlich. Es ist für eine Verbindung möglich, das Kriterium mehrerer Regeln zu erfüllen. Deshalb kann der PCK **286** Regeln in der Reihenfolge anwenden, die von dem Operator spezifiziert sind, wobei die Aktion der ersten passenden Regel genommen wird. Eine Default-Regel kann ebenfalls eingestellt werden, die die Aktion definiert, die eingesetzt werden soll für IP-Pakete, die auf keine der definierten Regeln passen. Der Satz von Regeln, die von dem Operator ausgewählt werden, kann in einem Pfadauswahlprofil definiert werden.

[0072] Beispielsweise kann eine Pfadauswahlregel den Pfad basierend auf jeder der nachfolgenden Pfadinformationen auswählen, in der IP-Pakete die Regel erfüllen: ein Hauptpfad, ein Zweitpfad und ein Drittpfad. Der Hauptpfad wird in jeder Auswahlregel spezifiziert. Der Zweitpfad wird nur verwendet, wenn der Hauptpfad ausgefallen ist. Falls der Zweitpfad nicht spezifiziert ist, können jegliche IP-Pakete, die die Regel erfüllen, verworfen werden, wenn der Hauptpfad ausgefallen ist. Der Drittpfad wird nur spezifiziert, falls ein Zweitpfad spezifiziert ist. Der Drittpfad wird ausgewählt, falls sowohl der Haupt- als auch der Zweitpfad ausgefallen sind. Falls kein Drittpfad spezifiziert ist, werden jegliche IP-Pakete, die die Regel erfüllen, verworfen, wenn sowohl der Haupt- als auch der Zweitpfad ausgefallen sind. Die Pfadauswahl kann verallgemeinert werden, derart, dass die Pfadauswahlregel bis zu N Pfade auswählen kann, wobei der N-te Pfad nur verwendet wird, falls der (N –

1)-te Pfad ausgefallen ist. Die zuvor angegebenen Beispiele, in denen N = 3 ist, sind rein beispielhaft, obgleich N typischerweise eine ziemlich kleine Zahl ist.

[0073] Beispielhaft wird der Betrieb des Systems **100** nachfolgend beschrieben. Zuerst wird eine Backbone-Verbindung zwischen den PEPs **200** der zwei Netzwerk-Gateways **120**, **140** (d.h. den zwei Spoofern) errichtet, die an jedem Ende der Backbone-Verbindung **130** angeordnet sind, für die das TCP-Spoofing gewünscht wird. Wann immer ein IP-Host **110** eine TCP-Verbindung initiiert, überprüft der TSK **280** des PEP **200**, der lokal zu dem IP Host **110** liegt, seine konfigurierten selektiven TCP Spoofing-Regeln. Falls die Regeln anzeigen, dass die Verbindung nicht gespooft werden soll, ermöglicht der PEP **200**, dass die TCP-Verbindung von Ende zu Ende ungespooft erfolgt. Falls die Regeln anzeigen, dass die Verbindung gespooft werden soll, antwortet der Spoofing PEP **200** lokal auf den TCP Drei-Wege-Handshake des IP Hosts. Parallel dazu sendet der Spoofing PEP **200** eine Nachricht über die Backbone-Verbindung **130** an sein Partner-Netzwerk-Gateway **140**, um ihn zu bitten, ein TCP Drei-Wege-Handshake mit dem IP Host **150** auf dessen Seite der Backbone-Verbindung **130** zu initiieren. Daten werden dann zwischen dem IP Host **110**, **150** mit dem PEP **200** des Netzwerk-Gateways **120** ausgetauscht, die empfangenen Daten werden lokal bestätigt und über die Backbone-Verbindung **130** über die Hochgeschwindigkeits-Backbone-Verbindung weitergeleitet, die Daten werden basierend auf den konfigurierten Komprimierungsregeln geeignet komprimiert. Die Priorität der TCP-Verbindung wird bestimmt, wenn die Verbindung aufgebaut ist. Der BPK **282** kann die Verbindung mit anderen empfangenen Verbindungen über eine einzelne Backbone-Verbindung multiplexen, der PK **284** bestimmt die Priorität der Verbindung, und der PSK **286** bestimmt den Pfad, den die Verbindung nehmen soll.

[0074] Der PEP **200**, wie zuvor beschrieben, verbessert vorteilhafterweise die Netzwerkleistung, indem TCP Spoofing-bezogene Quellen belegt werden, wie beispielsweise Pufferplatz, Steuerungsblöcke etc., nur für TCP-Verbindungen, für die Spoofing vorteilhaft ist; indem der Drei-Wege-Handshake gespooft wird, um die Datenantwortzeiten zu verringern; indem die Anzahl der ACKs reduziert wird, die übertragen werden, indem ein lokales Acknowledgement ausgeführt wird, und indem mehrere TCP-Verbindungen mit einem einzelnen ACK bestätigt werden; indem eine Datenkomprimierung erfolgt, um die Datenmenge zu erhöhen, die übertragen werden kann; indem Prioritäten zugewiesen werden auf unterschiedliche Verbindungen; und indem mehrere Pfade für herzustellende Verbindungen definiert werden.

[0075] Fig. 3 zeigt einen beispielhaften Stack, der das Verhältnis zwischen dem TCP Stack und den PEP-Kernel **280**, **282**, **284**, **286** der vorliegenden Erfindung darstellen. Der TSK **280** ist hauptsächlich verantwortlich für die Funktionen, die das TCP Spoofing betreffen. Der TSK **280** umfasst in einer beispielhaften Ausführungsform zwei Basiselemente: eine Transportschicht, die einen TCP Stack **303** und einen IP Stack **305** einschließt; und eine TCP Spoofing-Anwendung **301**. Die Transportschicht ist verantwortlich für die Interaktion mit den TCP Stacks (beispielsweise **303**) der IP Hosts **110**, die mit einer lokalen LAN-Schnittstelle **220** eines PEP **210** verbunden sind.

[0076] Der TSK **280** implementiert das TCP-Protokoll, das die geeigneten TCP-Zustandsmaschinen umfasst und die gespooften TCP-Verbindungen abschließt. Die TCP-Spoofing-Anwendung **301** sitzt auf der Transportschicht und wirkt als die Anwendung, die Daten von den Anwendungen des IP Host **110** empfängt und Daten zu diesem sendet. Aufgrund der Schichtarchitektur des Protokolls isoliert die TCP-Spoofing-Anwendung **301** die Details des TCP-Spoofing gegenüber der Transportschicht, so dass die Transportschicht in der Standardform arbeiten kann.

[0077] Wie in Fig. 3 gezeigt, kann die TCP-Spoofing-Anwendung **301** ebenfalls mit dem BPK **282** gekoppelt sein, der mit den WAN-Schnittstellen **230** verknüpft ist. Der BPK **282** führt eine Backbone-Protokoll-Aufrechterhaltung aus, und implementiert das Protokoll (oder Protokolle), mit dem die Netzwerke-Gateways **120**, **140** (in Fig. 1) kommunizieren. Der BPK **282** stellt eine zuverlässige Auslieferung von Daten bereit, verwendet eine relativ kleine Menge an Bestätigungsverkehr und unterstützt eine allgemeine Backbone-Verwendung (d.h. die Verwendung, die nicht spezifisch für TSK **280** ist); ein Beispiel eines Protokolls, das durch BPK **282** implementiert wird, ist das reliable data protocol (RDP; zuverlässige Datenprotokoll).

[0078] Der BPK **282** liegt entsprechend einer beispielhaften Ausführungsform über dem PK **284** und dem PSK **286**. Der PK **284** ist verantwortlich zur Festlegung der Priorität der IP-Pakete und dann zur Belegung der Übertragungsmöglichkeiten basierend auf der Priorität. Der PK **284** kann ebenfalls den Zugriff auf den Pufferraum steuern, indem die Warteschlangengröße gesteuert wird, die mit dem Senden und Empfang von IP-Paketen verknüpft ist. Der PSK **286** bestimmt, welchen Pfad ein IP-Paket nehmen soll, um sein Ziel zu erreichen. Der Pfad, der von dem PSK **286** ausgewählt ist, kann bestimmt werden, indem Pfadauswahlregeln angewendet werden. PSK **286** kann ebenfalls festlegen, welches IP-Paket weitergeleitet werden soll, indem ein alter-

nativer Pfad verwendet wird, und welche Pakete fallen gelassen werden sollen, wenn einer oder mehrere Pfade ausgefallen sind. Es ist festzustellen, das die zuvor erwähnte Anordnung rein beispielhaft ist; andere Anordnungen sind für einen Fachmann offenkundig.

[0079] Fig. 4A und Fig. 4B zeigen Flussdiagramme des Aufbaus einer gespooften TCP-Verbindung, in dem ein Drei-Wege-Handshake-Spoofing verwendet wird, bzw. ohne ein Drei-Wege-Handshake-Spoofing. Der TCP-Spoofing-Kernel **180** errichtet eine gespoofte TCP-Verbindung, wenn ein TCP <SYN>-Segment von dessen lokalem LAN empfangen wird oder eine Verbindungsanforderungsnachricht von dessen TSK-Netz. Es ist anzumerken, dass das Drei-Wege-Handshake-Spoofing gesperrt werden kann, um einen End-zu-End-Maximum-Segmentgrößen(MSS)-Austausch zu unterstützen, der nachfolgend deutlicher beschrieben werden wird. Zum Zwecke der Erläuterung wird der gespoofte TCP-Verbindungsaufbauprozess mit Bezug auf einen lokalen Host **400**, einen lokalen PIP-Endpunkt **402**, einen entfernten PEP-Endpunkt **404** und einen entfernten Host **406** beschrieben. Wie zuvor ausgeführt, liefert der TSK **280** innerhalb jedes PEP-Endpunkts **402** und **404** die Spoofing-Funktionalität.

[0080] In Schritt **401** sendet 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 es bereits einen Verbindungs-Steuerungsblock (CCB) gibt, der der TCP-Verbindung zugeordnet ist, die mit dem TCP-Segment verknüpft ist. Falls es keinen CCB gibt, prüft die Umgebung **402**, ob das TCP-Segment ein <SYN>-Segment ist, das zu einem nicht lokalen Ziel gesendet wurde. Falls dies der Fall ist, stellt das <SYN>-Segment einen Versuch dar, eine neue (nicht lokale) TCP-Verbindung aufzubauen, und die Umgebung **402** reicht das Segment an den TCP-Spoofing-Kernel **280** weiter, um die Disposition der TCP-Verbindung festzulegen. 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 die Verbindung gespooft werden soll. Falls die Verbindung gespooft werden soll, verwendet der TSK **280** (in einer beispielhaften Ausführungsform) die Priorität, die in dem ausgewählten TCP-Spoofing-Parameterprofil angegeben ist und den Partnerindex (der von der Umgebung **210** mit dem TCP <SYN>-Segment bereitgestellt wird), um den Handle der Backbone-Verbindung zu konstruieren, der verwendet werden sollte, um diese gespoofte TCP-Verbindung zu tragen. In der beispielhaften Ausführungsform wird der Partner-Index in den höheren 14 Bits des Handles angegeben, und die Priorität wird in den zwei niedrigeren Bits des Handles angegeben. Der Backbone-Verbindungs-Handle wird dann eingesetzt (über die TSK-Steuerungsblock (TCB)-Abbildungstabelle), um den TCB zu finden, der mit der Backbone-Verbindung verknüpft ist. Der TSK **280** des PEP-Endpunkts **402** überprüft dann, ob die Backbone-Verbindung steht. Falls die Backbone-Verbindung steht, bestimmt der TSK **280**, ob die Anzahl der gespooften TCP-Verbindungen, die bereits die ausgewählte Backbone-Verbindung verwenden, noch immer unter der TCP Verbindungssteuerungsblock (CCB) Ressourcengrenze liegt. Die CCB-Ressourcengrenze ist der kleinere Wert der lokalen Anzahl von CCBs (der als Parameter von der Plattformumgebung **210** bereitgestellt wird) und der Partner-Anzahl von CCBs (die in der neuesten TSK-Partnerparameter (TPP)-Nachricht von dem TSK-Partner empfangen wurde), die für diese Backbone-Verbindung verfügbar sind. Falls die Anzahl der Verbindungen noch unter der Grenze liegt, weist der TSK **280** des PEP-Endpunkts **402** einen eindeutigen TCP-Verbindungsidentifizierer (beispielsweise ein freier TCB-Abbildungstabelleneintragsindex) der Verbindung zu und ruft die Umgebung **210** auf, einen TCP-Verbindungssteuerungsblock für die Verbindung zu belegen.

[0081] Der TSK **280** des PEP-Endpunkts **402** gibt das TCP <SYN>-Segment zurück an die Umgebung **210**, das ungespooft weitergeleitet wird, falls einer der vorgenannten Überprüfungen fehlschlägt. Mit anderen Worten führen die folgenden Bedingungen dazu, dass die TCP-Verbindung ungespooft ist. Erstens, falls die selektiven TCP-Spoofing-Regeln anzeigen, dass die Verbindung nicht gespooft werden soll. Ebenfalls wenn es keine Backbone-Verbindung für die Priorität gibt, mit der die TCP-Verbindung gespooft werden soll (angezeigt durch Fehlen eines TCB für die Backbone-Verbindung). Kein Spoofing wird ausgeführt, falls die Backbone-Verbindung heruntergefahren ist. Zusätzlich, falls die Anzahl der gespooften TCP-Verbindungen, die bereits von der Backbone-Verbindung benutzt werden, einen vorbestimmten Schwellenwert erreicht oder überschreitet, wird dann kein Spoofing ausgeführt. Ferner, falls es keinen CCB-Verbindungstabelleneintrag gibt, der verfügbar ist oder es keinen CCB gibt, der von dem CCB-Freipool verfügbar ist, wird die TCP-Verbindung ungespooft weitergeleitet. Für den Fall, bei dem es keine Backbone-Verbindung gibt, kann der TSK **280** des PEP-Endpunkts **402** ebenfalls ein Ereignis absenden, um den Operator zu alarmieren, dass es eine Fehlanpassung zwischen den konfigurierten TCP-Spoofing-Parameterprofilen und dem konfigurierten Satz von Backbone-Verbindungen gibt.

[0082] Das Beispiel sei nun fortgeführt. Falls all die zuvor genannten Prüfungen bestanden werden, schreibt der TSK **280** des PEP-Endpunkts **402** den Backbone-Verbindungs-Handle in den Puffer, der das TCP

<SYN>-Segment hält. Es ist anzumerken, dass dies nicht getan wird, bis ein CCB von der Plattformumgebung **402** erfolgreich belegt wird, da die Umgebung den Puffer nicht zählt, solange nicht ein CCB erfolgreich belegt ist. TSK **280** kopiert dann die Parameter von dem ausgewählten TCP-Spoofing-Parameterprofil in den CCB. Folglich wird relevante Information (beispielsweise die maximale Segmentgröße, die von dem Host angezeigt wird (falls kleiner als der konfigurierte MSS), die anfängliche Reihenfolgennummer, etc.) aus dem TCB <SYN>-Segment kopiert und in dem TCB gespeichert. Es ist zu bemerken, dass die Quell- und Ziel-IP-Adressen und Quell- und Ziel-TCP-Portnummern bereits in den CCB von der Plattformumgebung **402** abgelegt wurden, als der CCB belegt wurde; die Umgebung **402** benutzt diese Information, um CCB-Hash-Funktion-Kollisionen zu verwalten.

[0083] Nach dem Belegen und Setzen des CCB baut der TCP-Spoofing-Kernel **280** des PEP-Endpunkts **402** eine Connection Request (CR)-Nachricht per Schritt **403** auf und sendet diese an sein TSK-Netz, das mit dem entfernten PEP-Endpunkt **404** verknüpft ist. Die CR-Nachricht enthält hauptsächlich all die Information, die aus dem TCP-Spoofing-Parameterprofil und dem TCP <SYN>-Segment extrahiert wurde, und wird in dem lokalen CCB gespeichert, beispielsweise die Quell- und Ziel-IP-Adressen, die Quell- und Ziel-TCP-Portnummern, der MSS-Wert, etc., mit Ausnahme der Felder, die nur lokale Bedeutung haben, wie beispielsweise die anfängliche Reihenfolgennummer. (Die IP-Adressen und die TCP-Portnummern werden in einen TCP-Verbindungsheader gesetzt.) Mit anderen Worten enthält die CR-Nachricht all die Information, welche der Partner-TSK des PEP-Endpunkts **404** für den Aufbau seiner eigenen CCB benötigt. Um den lokalen Verbindungsaufbau zu vervollständigen, sendet der TCP-Spoofing-Kernel **280** des lokalen PEP-Endpunkts **402** ein TCP <SYN, ACK>-Segment an den lokalen Host **400** in Antwort auf das <SYN>-Segment, das empfangen wurde in Schritt **405**. Der TSK **280** des PEP-Endpunkts **402** führt den Schritt **405** gleichzeitig mit dem Schritt des Sendens der Connection Request-Nachricht bzw. der Verbindungsanforderungsnachricht (d.h. Schritt **403**) aus, falls ein Drei-Wege-Handshake-Spoofing freigegeben ist. Anderenfalls wartet der TSK **280** von **402** auf eine Verbindungsaufbau(CE)-Nachricht bzw. Connection Established-Nachricht von seinem TSK-Partner des entfernten PEP-Endpunkts **404**, bevor das <SYN, ACK>-Segment gesendet wird. Bei einer beispielhaften Ausführungsform wählt der TSK **280** des PEP-Endpunkts **402** eine zufällige anfängliche Reihenfolgennummer aus (wie in IETF (Internet Engineering Task Force) RFC **793** bereitgestellt), um sie für das Senden von Daten zu verwenden.

[0084] Falls ein Drei-Wege-Handshake-Spoofing nicht freigegeben ist, wird der MSS-Wert, der in dem <SYN, ACK>-Segment gesendet wurde, auf den MSS-Wert gesetzt, der in der CE-Nachricht empfangen wurde. Falls das Drei-Wege-Handshake-Spoofing freigegeben ist, wird der MSS-Wert aus dem TCP-Spoofing-Parameterprofil bestimmt, das für die Verbindung ausgewählt wurde (und die konfigurierte Pfad-Maximum-Übertragungseinheit (MTU)). Für diesen Fall vergleicht dann der TSK **280** des PEP-Endpunkts **402** den MSS-Wert, der in der Connection Established-Nachricht empfangen wurde, wenn diese ankommt, mit dem Wert, der zu dem lokalen Host in dem TCP <SYN, ACK>-Segment gesendet wurde. Falls der MSS-Wert, der in der CE-Nachricht empfangen wurde, kleiner ist als der MSS-Wert, der von dem lokalen Host gesendet wurde, existiert eine Fehlanpassung einer maximalen Segmentgröße. (Falls eine MSS-Fehlanpassung existiert, muss der TSK die Größe der TCP-Datensegmente vor deren Senden einstellen). Nach dem Senden des TCP <SYN, ACK>-Segment (Schritt **405**) ist der TSK **280** des lokalen PEP-Endpunkts **402** bereit, um die Annahme von Daten von dem lokalen Host **400** zu starten. In Schritt **407** sendet der lokale Host **400** ein <ACK>-Segment an den TSK **280** des PEP-Endpunkts **402**; danach leitet der lokale Host in Schritt **409** ebenfalls Daten an den TSK **280** des PEP-Endpunkts **402**. Wenn das Drei-Wege-Handshake-Spoofing verwendet wird, muss der TSK **280** nicht auf die Connection Established-Nachricht warten, die von dessen TSK-Peer ankommt, bevor die Daten angenommen und weitergeleitet werden. Wie in **Fig. 4A** zu sehen, sendet 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** an den Partner-TSK des PEP-Endpunkts **404** (in Schritt **413**) vor dem Empfang einer CE-Nachricht von dem Partner-TSK des PEP-Endpunkts **404**.

[0085] Der TSK **280** des PEP-Endpunkts **402** akzeptiert jedoch keine Daten von seinem TSK-Partner des PEP-Endpunkts **404** bis nach dem die CE-Nachricht empfangen wurde. TSK **280** des PEP-Endpunkts **402** leitet jegliche Daten nicht weiter, die von seinem TSK-Partner des PEP-Endpunkts **404** empfangen wurden, zu dem 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**).

[0086] Wenn eine Connection Request-Nachricht von einem Partner-TSK (Schritt **403**) umfangen wurde, belegt der TCP-Spoofing-Kernel **280** ein CCB für die Verbindung und speichert dann alle relevanten Informationen aus der CR-Nachricht in dem CCB. TSK **280** des PEP-Endpunkts **404** benutzt dann diese Information, um ein TCP <SYN>-Segment zu erzeugen, wie in Schritt **415**, um dies zu dem entfernten Host **406** zu senden.

Das MSS in dem <SYN>-Segment wird auf den Wert gesetzt, der von dem TSK-Partner des PEP-Endpunkts **404** empfangen wurde. Wenn der entfernte Host mit einem TCP <SYN, ACK>-Segment antwortet (Schritt **417**), sendet der TSK **280** des PEP-Endpunkts **402** eine Connection Established-Nachricht an seinen TSK-Partner des entfernten PEP-Endpunkts **404** (Schritt **419**), die die CE-Nachricht des MSS enthält, die von dem lokalen Host in dem <SYN, ACK>-Segment gesendet wurde. Der TSK **280** des PEP-Endpunkts **402** antwortet ebenfalls als Schritt **421** mit einem TCP <ACK>-Segment, um den lokalen Drei-Wege-Handshake zu vervollständigen. Der Partner TSK des PEP-Endpunkts **404** leitet dann die Daten weiter, die vom TSK **280** empfangen wurden, an den Host, per Schritt **423**. Gleichzeitig sendet in Schritt **425** der entfernte Host **406** Daten an den Partner-TSK des PEP-Endpunkts **404**, der den Empfang der Daten bestätigt, indem ein <ACK>-Segment an den entfernten PEP-Endpunkt **404** abgegeben wird, per Schritt **427**. Gleichzeitig mit der Bestätigung werden die Daten an den TSK **280** des PEP-Endpunkts **402** gesendet (Schritt **429**).

[0087] An diesem Punkt ist TSK **280** bereit, Daten aus jeder Richtung zu empfangen und weiterzuleiten. TSK **280** leitet die Daten, wie in Schritt **431**, an den lokalen Host weiter, der wiederum ein <ACK>-Segment sendet (Schritt **433**). Falls die Daten von seinem TSK-Partner vor einer <SYN, ACK>-Segment Antwort von dem lokalen Host empfangen werden, werden die Daten in eine Warteschlange gebracht und dann gesendet, nachdem das <ACK>-Segment in Antwort auf das <SYN, ACK>-Segment gesendet wurde (wenn es ankommt).

[0088] Es sei nun auf Fig. 4B Bezug genommen. Eine gespoofte TCP-Verbindung wird aufgebaut, wobei das Drei-Wege-Handshake-Spoofing gesperrt bzw. nicht freigegeben ist. Bei diesem Szenario sendet der lokale Host **400** ein TCP <SYN>-Segment in Schritt **451** an den TSK **280** innerhalb des lokalen PEP-Endpunkts **402**. Im Gegensatz zu dem TCP-Verbindungsaufbau von Fig. 4A antwortet der lokale PEP-Endpunkt **402** nicht auf ein TCP <SYN>-Segment mit einem <SYN, ACK>-Segment, sondern leitet lediglich eine CR-Nachricht an den entfernten PEP-Endpunkt **404** (Schritt **453**). Als Nächstes in Schritt **455** sendet er ein TCP <SYN>-Segment an den entfernten Host **406**. In Antwort darauf übermittelt der entfernte Host **406** ein TCP <SYN, ACK>-Segment zurück an den entfernten PEP-Endpunkt **404** (in Schritt **457**). Danach leitet der entfernte PEP-Endpunkt **404** als Schritt **459** eine CE-Nachricht an den lokalen PEP-Endpunkt **402**, der darauf folgend ein <SYN, ACK>-Segment an den lokalen Host **400** in Schritt **461** abgibt. Gleichzeitig mit Schritt **459** gibt der entfernte PEP-Endpunkt **404** ein <ACK>-Segment an den entfernten Host **406** (Schritt **463**) ab.

[0089] Beim Empfang des <ACK>-Segments kann der entfernte Host **406** mit der Datenübertragung beginnen, als Schritt **465**. Sobald der PEP-Endpunkt **404** die Daten von dem entfernten Host **406** empfängt, überträgt gleichzeitig der entfernte PEP-Endpunkt **404** in Schritt **467** die TD-Nachricht an den lokalen PEP-Endpunkt **402** und sendet ein <ACK>-Segment an den entfernten Host **406**, um den Empfang der Daten zu bestätigen (Schritt **469**).

[0090] Da der lokale Host **400** ein <SYN, ACK>-Segment von dem lokalen PEP-Endpunkt **402** empfangen hat, bestätigt der lokale Host **400** die Nachricht in Schritt **471**. Danach sendet der lokale Host **400** Daten an den lokalen PEP-Endpunkt **402**. Bevor der lokale PEP-Endpunkt **402** in diesem Beispiel die Daten von dem lokalen Host **400** empfängt, leitet der lokale PEP-Endpunkt **402** die Daten weiter, die von dem entfernten Host **406** stammen, über die TD-Nachricht (Schritt **467**) an den lokalen Host **400**, als Schritt **475**.

[0091] In Antwort auf die empfangenen Daten (in Schritt **473**) gibt der lokale PEP-Endpunkt **402** ein <ACK>-Segment als Schritt **477** ab und leitet die Daten in einer TD-Nachricht an den entfernten PEP-Endpunkt **404** als Schritt **479** weiter. Der lokale Host **400** antwortet auf die empfangenen Daten von Schritt **475** mit einem <ACK>-Segment zu dem lokalen PEP-Endpunkt **402** (Schritt **481**). Der entfernte PEP-Endpunkt **404** sendet die Daten von dem lokalen Host **400** als Schritt **483** beim Empfang der TD-Nachricht. Nach Empfang der Daten bestätigt der entfernte Host **406** den Empfang durch Senden eines <ACK>-Segments zurück an den entfernten PEP-Endpunkt **404**, als Schritt **485**.

[0092] Fig. 5 zeigt den Strom der Pakete mit der PEP-Architektur entsprechend einer Ausführungsform der vorliegenden Erfindung. Wie gezeigt umfasst ein Kommunikationssystem **500** einen PEP-Endpunkt **501** auf einer Hub-Seite (oder lokal), der mit einem PEP-Endpunkt **503** an einem entfernten Ort über eine Backbone-Verbindung verbunden ist. Auf der Hub-Seite bzw. dem Hub-Ort (oder dem lokalen Ort) und an jedem entfernten Ort handhaben die PEP-Endpunkte **501** und **503** beispielhaft IP-Pakete. PEP-Endpunkt **501** umfasst ein internes IP-Paket Routingmodul **501a**, das lokale IP-Pakete empfängt und diese Pakete mit einem TSK **501b** und einem BPK **501c** austauscht. In gleicher Weise umfasst der entfernte PEP-Endpunkt **503** ein internes IP-Paket Routingmodul **503a**, dessen Kommunikation mit einem TSK **503b** und einem BPK **503c** steht. Mit Ausnahme der Tatsache, dass der PEP-Endpunkt **501** auf der Hub-Seite sehr viel mehr Backbone-Protokoll-Verbindungen unterstützen kann als ein PEP-Endpunkt **503** auf der entfernten Seite, sind die PEP-Verarbeitungen auf der

Hub- und der entfernten Seite symmetrisch.

[0093] Für einen Verkehr von lokal zu WAN (d.h. Upstream-Richtung) empfängt der PEP-Endpunkt **501** IP-Pakete von seiner lokalen Schnittstelle **220** (Fig. 2). Nicht-TCP-IP-Pakete werden zu der WAN-Schnittstelle **230** (Fig. 2) weitergeleitet (wenn gewünscht). TCP-IP-Pakete werden intern zur TSK **501b** weitergeleitet. TCP-Segmente, die zu Verbindungen gehören, die nicht gespooft werden, werden von dem Spoofing-Kernel **501b** zu dem Routingmodul **501a** gereicht, um unverändert zu der WAN-Schnittstelle **230** geleitet zu werden. Für gespoofte TCP-Verbindungen schließt der TCP-Spoofing-Kernel **501a** lokal die schließt der TCP-Spoofing-Kernel **501a** lokal die TCP-Verbindung ab. TCP-Daten, die von einer gespooften Verbindung empfangen werden, werden von dem Spoofing-Kernel **501a** zu dem Backbone-Protokoll-Kernel **501c** gereicht und dann auf die geeignete Backbone-Protokoll-Verbindung gemultiplext. Der Backbone-Protokoll-Kernel **501c** gewährleistet, dass die Daten über das WAN ausgeliefert werden.

[0094] Bei einem Verkehr von WAN zu lokal (d.h. Downstream-Richtung) 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 an die lokale Schnittstelle **220** (Fig. 2) (geeignet) weitergeleitet. IP-Pakete, die für den Endpunkt **503** adressiert sind, die einen Next Protocol Header-Typ von "PBP" haben, werden zu dem Backbone-Protokoll-Kernel **503c** geleitet. Der Backbone-Protokoll-Kernel **503c** extrahiert die TCP-Daten und leitet sie an den TCB-Spoofing-Kernel **503b** zur Übertragung auf die geeignete gespoofte TCP-Verbindung weiter. Zusätzlich zum Übertragen der TCP-Daten wird die Backbone-Protokoll-Verbindung von dem TCP-Spoofing-Kernel **501b** benutzt, um Kontrollinformation zu seinem Partner-TCP-Spoofing-Kernel **503b** in dem entfernten PEP-Endpunkt **503** zu senden, um den Verbindungsaufbau und den Verbindungsabbruch zu koordinieren.

[0095] Eine Priorisierung kann an vier Punkten an dem System **500** angewendet werden innerhalb des Routingmoduls **501a** und dem TSK **501b** des PEP-Endpunkts **501** und innerhalb des Routingmoduls **503a** und dem TSK **503b** des PEP-Endpunkts **503**. In Upstream-Richtung werden Prioritätsregeln auf die Pakete der einzelnen TCP-Verbindungen am Eintrittspunkt zu dem TCP-Spoofing-Kernel **501** angewendet. Diese Regeln ermöglichen es einem Kunden, zu steuern, welche gespooften Anwendungen höhere und niederere Prioritätszugriffe auf die Spoofing-Ressourcen haben. Eine Upstream-Priorisierung wird ebenfalls vor dem Weiterleiten der Pakete zu dem WAN angewendet. Dies ermöglicht es einem Kunden, die relative Priorität der gespooften TCP-Verbindungen mit Bezug auf die ungespooften TCP-Verbindungen und Nicht-TCP-Verkehr zu steuern (sowie die relative Priorität dieser anderen Typen von Verkehr mit Bezug aufeinander zu steuern). Auf der Downstream-Seite wird die Priorisierung verwendet, um den Zugriff auf den Pufferplatz zu steuern und auf andere Ressourcen in dem PEP-Endpunkt **503** allgemein und mit Bezug auf das TCP-Spoofing.

[0096] Auf der Hub- (oder lokalen) Seite kann der PEP-Endpunkt **501** in einem Netzwerk-Gateway (beispielsweise ein IP-Gateway) implementiert sein entsprechend einer Ausführungsform der vorliegenden Erfindung. Auf der entfernten Seite kann der PEP-Endpunkt **503** in der Komponente der entfernten Seite implementiert sein, beispielsweise einem Satellitenterminal, wie beispielsweise einem Multimedia-Relay, Multimedia-VSAT oder einer Personal Earth Station (PES) Remote.

[0097] Die Architektur des Systems **500** stellt eine Anzahl von Vorteilen bereit. Zunächst kann ein TCP-Spoofing sowohl in Upstream- als auch in Downstream-Richtung erreicht werden. Zusätzlich unterstützt das System ein Spoofing des TCP-Verbindungsstarts, und ein selektives TCP-Spoofing nur mit Verbindungen, die vom Spoofing profitieren können. Ferner ermöglicht das System **500** eine Priorisierung der gespooften TCP-Verbindungen für einen Zugriff auf die TCP-Spoofing-Ressourcen (beispielsweise verfügbare Bandbreite und Pufferplatz). Diese Priorisierung wird für alle Typen von Verkehr verwendet, die nach Systemressourcen ringen.

[0098] Mit Bezug auf die Backbone-Verbindung ist das System **500** zur Anwendung in einem Satellitennetzwerk als WAN geeignet. Das heißt, dass das Backbone-Protokoll für Satellitenverwendung optimiert ist, indem Kontrollblockressourcenanforderungen minimiert werden und eine effiziente Fehlerentdeckung für verloren gegangene Pakete bereitstellt. Das System **500** liefert ebenfalls einen Feedback-Mechanismus, um maximale Pufferplatzressourceneffizienz zu unterstützen. Ferner stellt das System **500** einen reduzierten Bestätigungsverkehr bereit, indem ein einzelnes Backbone-Protokoll ACK verwendet wird, um die Daten mehrerer TCP-Verbindungen zu bestätigen.

[0099] Fig. 6 zeigt den Fluss der IP-Pakete durch einen PEP-Endpunkt entsprechend einer Ausführungsform der vorliegenden Erfindung. Wenn IP-Pakete in der lokalen LAN-Schnittstelle **220** empfangen werden, bestimmt der PEP-Endpunkt **210** (wie durch den Entscheidungspunkt A gezeigt), ob die Pakete für einen Host

bestimmt sind, der lokal liegt; falls dies der Fall ist, werden die IP-Pakete zu der passenden lokalen LAN-Schnittstelle **220** geleitet. Falls die IP-Pakete für einen entfernten Host bestimmt sind, entscheidet dann der PEP-Endpunkt **210** am Entscheidungspunkt B, ob der Verkehr ein TCP-Segment ist. Falls der PEP-Endpunkt **210** festlegt, dass die Pakete tatsächlich TCP-Segmente sind, bestimmt dann der TSK **280**, ob die TCP-Verbindung gespooft werden sollte. Falls jedoch der PEP-Endpunkt **210** bestimmt, dass die Pakete keine TCP-Segmente sind, verarbeitet dann der BPK **282** den Verkehr zusammen mit dem PK **284** und dem PSK **286** für eine mögliche Übertragung in das WAN. Es ist zu bemerken, dass der BPK **282** keine ungespooften IP-Pakete verarbeitet; d.h., dass die Pakete direkt zum PK **284** strömen. Wie in Fig. 6 zu sehen, wird der Verkehr, der von der WAN-Schnittstelle **230** empfangen wird, geprüft, um zu bestimmen, ob der Verkehr ein richtiges PBP-Segment (Entscheidungspunkt D) für den bestimmten PEP-Endpunkt **210** ist; falls die Bestimmung dies bestätigt, werden dann die Pakete zu dem BPK **282** und dann dem TSK **280** gesendet.

[0100] Eine Routing-Unterstützung umfasst das Routen zwischen den Ports des PEP-Endpunkts **210** (Fig. 2), beispielsweise von einem Multimedia VSAT LAN-Port zu einem anderen. Hinsichtlich der Architektur passen die Funktionalitäten des TCP-Spoofings, der Priorisierung und der Pfadauswahl zwischen die IP-Routing-Funktionalität und das WAN. Die PEP-Funktionalität muss nicht auf IP-Pakete angewendet werden, die von einem lokalen Port zu einem lokalen Port innerhalb des gleichen PEP-Endpunkts **210** geroutet werden. TCP-Spoofing, Priorisierung und Pfadauswahl werden auf IP-Pakete angewendet, die von einer lokalen PEP-Endpunkt-Schnittstelle empfangen werden, die dazu bestimmt wurden, für eine andere Seite bzw. einen anderen Ort als Ziel zu dienen, durch die Routingfunktion.

[0101] Fig. 7 zeigt das Verhältnis zwischen den PEP-Endpunkten und den PEP-Endpunkt-Profilen entsprechend einer Ausführungsform der vorliegenden Erfindung. PEP-Parameter sind hauptsächlich konfiguriert über eine Menge von Profilen **701** und **703**, die mit einem oder mehreren PEP-Endpunkten **705** verknüpft sind. Bei einer beispielhaften Ausführungsform sind die PEP-Parameter konfiguriert auf einer PEP-Endpunkt-Basis, so als ob TCP-Spoofing global freigegeben ist. Diese Parameter werden in den PEP-Endpunkt-Profilen **701** und **703** konfiguriert. Es ist anzumerken, dass Parameter, die auf spezifische PEP-Kernels angewendet werden, konfiguriert werden können über andere Typen von Profilen. Profile **701** und **703** sind eine Netzwerkmanagementkonstruktion; intern verarbeitet ein PEP-Endpunkt **705** einen Satz von Parametern, die über ein oder mehrere Files empfangen werden.

[0102] Wann immer der PEP-Endpunkt **705** neue Parameter empfängt, vergleicht die Plattformumgebung die neuen Parameter mit den vorhandenen Parametern, findet heraus, ob die PEP-Kernel durch die Parameteränderungen betroffen sind und reicht dann die neuen Parameter an die betroffenen Kernel. Bei einer beispielhaften Ausführungsform sind alle Parameter dynamisch installiert. Mit Ausnahme der Parameter, die komponentenspezifisch sind (wie beispielsweise die IP-Adressen einer Komponente) können alle Parameter mit Default-Werten definiert werden.

[0103] Wie zuvor erwähnt, kann der PEP-Endpunkt **210** in einer Anzahl von unterschiedlichen Plattformen implementiert sein, entsprechend den verschiedenen Ausführungsformen der vorliegenden Erfindung. Diese Plattformen können ein IP-Gateway, ein Multimedia Relay, ein Multimedia VSAT (Very Small Aperture Terminal) (Terminal mit sehr kleiner Antennenapertur) und ein Personal Earth Station (PES) Remote umfassen, wie in Fig. 8 bis Fig. 11 gezeigt. Allgemein und wie in Fig. 2 diskutiert, definiert der PEP-Endpunkt **210** eine lokale LAN-Schnittstelle **220**, eine Schnittstelle, durch die der PEP-Endpunkt **210** mit IP-Hosts verbunden ist, die an dem Ort platziert sind. Eine WAN-Schnittstelle **230** ist eine Schnittstelle, über die der PEP-Endpunkt **210** mit anderen Orten verbunden ist. Es ist anzumerken, dass eine WAN-Schnittstelle **230** physisch ein LAN-Port sein kann.

[0104] Fig. 8 bis Fig. 11 beschreiben nachfolgend die spezifischen LAN- und WAN-Schnittstellen der verschiedenen spezifischen PEP-Endpunkt-Plattformen. Die bestimmten LAN- und WAN-Schnittstellen, die verwendet werden, hängen ab von den verwendeten PEP-Endpunkten am entfernten Ort, von der Konfiguration des Hubs und der PEP-Endpunkte am entfernten Ort und von den Pfadauswahlregeln, die konfiguriert sein können.

[0105] Fig. 8 zeigt die Schnittstellen des PEP-Endpunkts, der als IP-Gateway implementiert ist, entsprechend einer Ausführungsform der vorliegenden Erfindung. Beispielhaft hat ein IP-Gateway **801** eine einzelne lokale LAN-Schnittstelle, die eine Unternehmensschnittstelle **803** ist. Das IP-Gateway **803** verwendet zwei WAN-Schnittstellen **805** zum Senden und Empfangen von IP-Paketen zu und von den PEP-Endpunkten am entfernten Ort: Eine Backbone-LAN-Schnittstelle und eine wide area access (WAA)-LAN-Schnittstelle.

[0106] Die Backbone-LAN-Schnittstelle **805** wird verwendet, um IP-Pakete zu PEP-Endpunkten am entfernten Ort zu senden über beispielsweise ein Satelliten-Gateway (SGW) und eine VSAT-Outroute. Eine VSAT-Outroute kann direkt von Multimedia Relays (Fig. 9) und Multimedia VSATs (Fig. 10) empfangen werden (und ist der Hauptpfad, der mit diesen Endpunkten verwendet wird); IP-Pakete können jedoch zu einem PES Remote (Fig. 11) über eine VSAT-Outroute gesendet werden.

[0107] Fig. 9 zeigt eine Multimedia Relay-Implementierung eines PEP-Endpunkts entsprechend einer Ausführungsform der vorliegenden Erfindung. Ein Multimedia Relay hat zwei oder drei lokale LAN-Schnittstellen **903**. Ein Multimedia Relay **901** hat bis zu zwei WAN-Schnittstellen **905** zum Senden von IP-Paketen zu den PEP-Endpunkten auf der Hub-Seite: Eine seiner LAN-Schnittstellen und eine serielle PPP-Schnittstelle, und vier oder fünf Schnittstellen zum Empfang von IP-Paketen von den PEP-Endpunkten auf der Hub-Seite, einer VSAT Outroute, alle LAN-Schnittstellen und eine serielle PPP-Schnittstelle. Es ist anzumerken, dass eine serielle PPP(Punkt-zu-Punkt-Protokoll)-Schnittstelle und eine LAN-Schnittstelle allgemein nicht gleichzeitig benutzt werden sollen.

[0108] Ein Multimedia Relay **901** unterstützt die Verwendung aller LAN-Schnittstellen **903** zur gleichen Zeit, um IP-Pakete zu und von den PEP-Endpunkten auf der Hub-Seite zu senden und zu empfangen. Ferner unterstützt ein Multimedia Relay **905** die Verwendung einer VADB (VPN Automatic Dial Backup) seriellen Schnittstelle zum Senden und Empfangen von IP-Paketen zu und von den PEP-Endpunkten auf der Hub-Seite.

[0109] Fig. 10 zeigt eine Multimedia VSAT-Implementierung des PEP-Endpunkts entsprechend einer Ausführungsform der vorliegenden Erfindung. Ein Multimedia VSAT **1001** in einer beispielhaften Ausführungsform besitzt zwei lokale LAN-Schnittstellen **1003**. Die Unterstützung für eine oder mehrere lokale serielle PPP-Schnittstellen kann verwendet werden. Der Multimedia VSAT **1001** besitzt zwei WAN-Schnittstellen **1005** zum Senden von IP-Paketen zu den PEP-Endpunkten auf der Hub-Seite: Eine VSAT inroute und eine der LAN-Schnittstellen. Die Multimedia VSAT **1001** hat damit drei Schnittstellen zum Empfang von IP-Paketen von den PEP-Endpunkten auf der Hub-Seite, der VSAT outroute und beiden LAN-Schnittstellen **1003**. Eine Multimedia VSAT **1003** kann die Verwendung von beiden LAN-Schnittstellen **1003** zur gleichen Zeit unterstützen, um IP-Pakete zu und von den PEP-Endpunkten auf der Hub-Seite zu senden und zu empfangen. Der Multimedia VSAT **1003** unterstützt ferner die Verwendung einer VADB-seriellen Schnittstelle zum Senden und Empfangen von IP-Paketen zu und von den PEP-Endpunkten auf der Hub-Seite.

[0110] Fig. 11 zeigt eine PES-Remote- bzw. -Fernimplementierung eines PEP-Endpunkts entsprechend einer Ausführungsform der vorliegenden Erfindung. Ein PES-Remote **101** kann eine lokale LAN-Schnittstelle und/oder mehrere lokale IP (beispielsweise PPP, SLIP, etc.) serielle Portschnittstellen aufweisen, die zusammen als LAN-Schnittstellen **1103** bezeichnet sind. Die einzelnen LAN-Schnittstellen **1103** hängen von der spezifischen PES-Remote-Plattform ab. Die PES-Remote **1101** hat in einer beispielhaften Ausführungsform bis zu fünf WAN-Schnittstellen **1105**, um IP-Pakete zu den PEP-Endpunkten auf der Hubseite, einem ISBN-Inroute, einer LAN-Schnittstelle, einer VADB seriellen Portschnittstelle, einer Frame Relay seriellen Portschnittstelle und einer IP-seriellen Portschnittstelle zu senden, und bis zu fünf existierende Schnittstellen zum Empfangen von IP-Paketen von den PEP-Endpunkten auf der Hubseite, einem ISBM-Outroute, einer LAN-Schnittstelle, einer VADB seriellen Portschnittstelle, einer Frame-Relay seriellen Portschnittstelle und einer IP-seriellen Portschnittstelle. Die physikalische Frame-Relay serielle Portschnittstelle kann mehrere Permanent-Virtual-Circuits (PVCs; permanente virtuelle Schaltungen) unterstützen; einige von diesen sind gleich den lokalen Schnittstellen **1103** und einige von ihnen sind WAN-Schnittstellen **1105**.

[0111] Der Backbone-Protokoll-Kernel **282** wird nachfolgend beschrieben. Es sei angemerkt, dass, obgleich der BPK **282** im Hinblick auf Interaktion mit dem TCP-Spoofing-Kernel **280** diskutiert wird, der BPK **282** allgemein genug ist, um die Verwendung von PBP Backbone-Verbindungen für andere Anwendungen als TCP-Spoofing zu unterstützen. Ein Beispiel einer solchen Anwendung ist die Unterstützung der Verwendung eines Stateful-Compression-Algorithmus für den Backbone-Link. Stateful-Compression-Algorithmen erfordern allgemein fehlerfreie Verbindungen. Ein Backbone-Protokoll kann verwendet werden, um eine Verbindung in eine fehlerfreie Verbindung zu überführen, die andernfalls aufgrund von Bitfehlern Pakete verlieren würde.

[0112] BPK **282** Parameter können über Profile konfiguriert werden. BPK **282** Parameter können in PEP-Endpunktprofilen definiert werden. Beispielhafte PEP-Endpunktprofile **701**, **703** sind zuvor in Verbindung mit Fig. 7 beschrieben. Welches PEP-Endpunkt-Profil **701**, **703** von einem PEP-Endpunkt **705** eingesetzt werden soll kann als Teil einer individuellen spezifischen Konfiguration des PEP-Endpunkts konfiguriert werden.

[0113] Profile sind ein Netzwerkverwaltungs- bzw. Managementkonstrukt. BPK **282** kann seine Parameter als

Datenstruktur empfangen, die von der Plattformumgebung **210** zu dem BPK **282** geführt werden. Parameter, die den BPK **282** selbst betreffen und alle Backbone-Verbindungen können auf einmal spezifiziert werden. Parameter, die einzelne Backbone-Verbindungen betreffen, können dem BPK **282** von der Plattformumgebung bereitgestellt werden, wenn die Verbindung geöffnet wird. Die Plattformumgebung **210** kann wiederum alle Parameter über Files empfangen, die über einen Netzwerkmanager zu ihr gesendet werden.

[0114] Der BPK **282** kann Parameter von der Plattformumgebung **210** beim Start empfangen, und wann immer die Plattformumgebung neue Parameter empfängt, die Änderungen der BPK **282** betreffenden Parameter enthalten. Wenn BPK **282** neue Parameter empfängt, kann es die neuen Parameter mit den existierenden Parametern vergleichen und dann eine Aktion ausführen, um die neuen Parameter zu installieren auf der Basis jener Parameter, die geändert wurden. Allgemein können alle Parameter dynamisch installiert werden. In einigen Fällen können Parameteränderungen einen Neustart der Backbone-Verbindung erforderlich machen.

[0115] Der PBPK **282** benutzt einen PBP Backbone-Verbindungssteuerungsblock (BCB), um Information betreffend eine bestimmte PBP Backbone-Verbindung zu speichern. Eine BCB-Abbildungstabelle **1200** wird von dem BPK **282** verwendet, um Zeiger **1202** auf BCBs **1204** zu speichern. Fig. 12 zeigt die Verwendung der BCB-Abbildungstabelle **1200**. BPK **282** benutzt den Backbone-Verbindungshandle, der der Plattformumgebung **210** als Index in die Abbildungstabelle zugeordnet ist. In einer beispielhaften Ausführungsform wird der Handle **1206** der BPK **282** über die Plattformumgebung **210** und durch den TSK **280** zugeführt, wann immer eine Prozedur aufgerufen wird, die auf die Backbone-Verbindung referenziert (beispielsweise wenn die TSK **280** eine TSK-Nachricht an BPK **282** leitet, um über die Backbone-Verbindung zu übertragen). Der Handle wird als Ziel-Steuerungsblock-Identifier (CBID) **1208** in den PBP-Segmenten verwendet, die von einem WAN empfangen werden, wobei der BPK **282** den Identifier lernt, der von seinen Partnern für die Verbindung während des Verbindungsaufbaus verwendet wird. (Ein Wert von 0×FFFF kann als Ziel CBID **1208** verwendet werden, bis der BPK **282** den CBID des Partners für die Verbindung lernt.

[0116] Falls ein PEP-Endpunkt **705** eine Ziel-CBID **1208** von 0×FFFF in einem PBP-Segment empfängt, kann er den Backbone-Verbindungshandle **1208** aus dem Partnerindex aufbauen, der mit der Quelle des PBP-Segments und der Portnummer in dem PBP-Header verknüpft ist.) Falls das einzelne Backbone-Protokoll, das verwendet wird, nicht direkt die Verwendung eines CBID als „Adress"-Feld in dem Protokoll erlaubt, kann der BPK entweder den CBID in der „Adresse" eingebettet haben, falls die „Adresse" mehr Bits als ein CBID besitzt, oder kann eine einfache Abbildungstabelle benutzen, um zwischen den CBIDs, und welchem geforderten Typ von „Adresse" auch immer, zu übersetzen falls die „Adresse" weniger Bits als eine CBID besitzt oder das „Adress"-Format nicht das Einbetten der CBID in die „Adresse" unterstützt. („Adresse" bezieht sich auf irgendein Feld, das von dem Protokoll verwendet wird, um unterschiedliche Verbindungen zu unterscheiden. Falls beispielsweise das verwendete Protokoll das TCP ist, ist das „Adress"-Feld die TCP-Portnummer.)

[0117] Tabelle 1 zeigt ein beispielhaftes BCB **1204**.

Tabelle 1

```
/* BP per connection stats should mirror total stats in bpk_stats_t */
typedef struct bpk_conn_stats_struct
{
   word  seq_q_pkts;      /* pkts in sequential queue */
   word  reseq_q_pkts;    /* pkts in ressequencing queue */

   /* counters which may be cleared */
   struct
   {
      /* general counters */
      word  tx_pkt;          /* BP pkts transmitted */
      word  rx_pkt;          /* BP pkts received */
```

```
        word   tx_poll;          /* POLLs transmitted */
        word   rx_poll;          /* POLLs received */
        word   tx_sack;          /* SACKs transmitted */
        word   rx_sack;          /* SACKs received */
        word   num_state_open;   /* times conn state changed to Open */

        /* error counters to debug a problem */
        word   tx_syn;           /* SYNs transmitted */
        word   rx_syn;           /* SYNs received */
        word   tx_rst;           /* RSTs transmitted */
        word   rx_rst;           /* RSTs received */
        word   retx;             /* retransmissions */
        word   bcb_mismatch;     /* pkts not matching indicated BCB */
        word   rx_dup_syn;       /* duplicate SYNs received */
        word   rx_outside_win;   /* pkts outside receive window */
        word   flush_seq_q_pkts; /* sequential queue pkts flushed */
        word   flush_reseq_q_pkts; /* resequencing queue pkts flushed */
        word   syn_fail;         /* times 3-way handshake failed */
        word   syn_restart;      /* SYN recevied when Open */
        word   peer_no_reply;    /* no reply from peer */
        word   internal_err;     /* unexpected internal error */
    } ctr;
} bpk_conn_stats_t;

/* All application users of BPK must provide a generic callback for data */
typedef void (* data_callback_t)(word cbid, pe_buffer_t *buf_ptr, word
pdu_len);

typedef struct bcb_struct
{
    word        scbid;      /* our CBID */
    word        dcbid; /* peer's CBID */
```

```
    dword           src_ip;/* our IP address */
    dword           dst_ip;/* peer's IP address */
    byte            port;          /* BP port */
    pbps_t          state;
    byte            flags;
    byte            poll_retry;
    dword           poll_timer;
    dword           ack_timer;
    word            irs;           /* Initial Receive Sequence number */
    word            snd_next;
    word            snd_una;
    word            snd_max;       /* Tx window */
    word            rcv_max;       /* Rx window */
    word            rcv_next;      /* next in-sequence expecting */
    word            last_rcv_next;
    data_callback_t data_callback;
    bpk_buffer_t    *seq_head;
    bpk_buffer_t    *seq_tail;
    bpk_buffer_t    *past_win;
    bpk_buffer_t    *retx_head;
    bpk_buffer_t    *retx_tail;
    bpk_buffer_t    *reseq_head;

    bpk_conn_stats_t stats;
} bcb_t;
```

[0118] Ein Operator kann die Backbone-Verbindungen für jeden PEP-Endpunkt **705** konfigurieren. Eine separate Verbindung wird für jede gewünschte Priorität konfiguriert. Beim Start fordert die Plattformumgebung **210** den BPK **282** auf, jede der konfigurierten Verbindungen des Endpunkts aufzubauen. Eine Backbone-Verbindung kann entweder passiv (in diesem Fall wartet BPK **282** auf seinen Partner, um den Start der Verbindung zu versuchen) oder aktiv (in diesem Fall versucht BPK **282**, die Verbindung zu starten) sein. Beide PEP-Endpunkte **705** in einer Verbindung können alle Backbone-Verbindungen als aktiv öffnen. Dies macht den Aufbau von Backbone-Verbindung sehr aggressiv, unterstützt eine schnelle Wiederherstellung von verschiedenen Fehlerszenarien (wie beispielsweise IP-Gateway-Redundanzswitches), obgleich auf Kosten der Benutzung von zusätzlicher Bandbreite, wenn beide PEP-Endpunkte zur gleichen Zeit gestartet werden.

[0119] Während spezifische Details von Protokoll zu Protokoll leicht variieren können, beschreibt das Folgende allgemein die Funktionsweise eines Backbone-Protokolls, das von der Erfindung benutzt wird.

[0120] Wenn als aktiv geöffnet baut BPK **282** eine Backbone-Verbindung auf, indem ein Handshake verwendet wird, der sehr ähnlich zu dem Drei-Wege-Handshake ist, der vom TCP verwendet wird (in Fig. 15 dargestellt). BPK **282** sendet ein BP-<SYN>-Segment (oder Ähnliches) an seinen BPK **282** Partner. Der Partner BPK

282 antwortet auf ein empfangenes BP-<SYN>-Segment mit einem BP-<SYN, ACK>-Segment (oder Ähnlichem). Wenn BPK **282** das <SYN, ACK>-Segment empfängt, antwortet es mit einem BP-<ACK>-Segment (oder Ähnlichem) und die Verbindung ist bereit, Backbone-Verkehr (beispielsweise TSK **280** Nachrichten) zu tragen.

[0121] Für jede Verbindung ruft die BPK **282** die Plattformumgebung **210** auf, um anzuzeigen, dass die Verbindung aktiv ist (d.h. offen), wenn sie ein <SYN, ACK>-Segment in Antwort auf ein übertragenes <SYN>-Segment empfängt oder ein <ACK>-Segment in Antwort auf ein übertragenes <SYN, ACK>-Segment.

[0122] Falls die Plattformumgebung **210** keine Verbindung geöffnet hat, wird der Partner BPK **282** nicht auf das <SYN>-Segment antworten. Wenn dies geschieht, wird der BPK periodisch ein neues <SYN>-Segment senden, um die Verbindung aufzubauen. BPK wird weiter versuchen, die Verbindung aufzubauen, bis sie entweder aufgebaut ist oder die Plattformumgebung **210** die Verbindung schließt.

[0123] Der BPK **282** kann zur Aufrechterhaltung aller Backbone-Verbindungen verantwortlich sein. Falls eine Verbindung abbricht, kann der BPK **282** eine Statusänderung an die Plattformumgebung **210** berichten und versuchen, die Verbindung wieder aufzubauen (die Plattformumgebung **210** benachrichtigt wiederum den Benutzer/die Benutzer der Backbone-Verbindung, beispielsweise TSK **280**.) Wenn die Verbindung wieder steht, wird eine Statusänderung wieder an die Plattformumgebung **210** berichtet. Während Perioden, in denen keine TSK-Nachrichtenaktivität vorliegt, kann BPK **282** einen Inaktivitätstimer für jede Verbindung laufen lassen. Wann immer der Timer ausläuft, kann BPK **282** ein BP-<NUL>-Segment (oder Ähnliches) an seinen BPK-Partner senden, um sicherzustellen, dass die Backbone-Verbindungsleitung noch aktiv ist. Um unnötigen Verbrauch von Bandbreite zu vermeiden, sollte ein Inaktivitätstimer, der von einem BPK **282** verwendet wird, länger sein (um zumindest eine Gesamtlaufzeit) als der Inaktivitätstimer des anderen BPK **282**. Dies kann beide BPK **282** Partner daran hindern, BP-<NUL>-Pakete gleichzeitig einander zuzusenden.

[0124] Das PEP-Backbone-Protokoll kann eine Prüfsumme umfassen, die verwendet werden kann, um Fehler in einem BP-Segment zu erfassen. Da Fehler selten sind, kann die Prüfsumme standardmäßig abgeschaltet werden, um CPU zu sparen. (Einige Backbone-Protokolle unterstützen nicht die Fähigkeit, die Prüfsumme abzuschalten.) Die Prüfsumme kann wieder von dem Operator freigegeben bzw. eingeschaltet werden, falls der Operator glaubt, dass Verfälschungsfehler auftreten. (BP-Prüfsummen können für eine oder beide Richtungen einer Verbindung eingeschaltet werden.) Da das Berechnen einer BP-Prüfsumme den Zugang zu dem gesamten BP-Segment erfordert, ist das Berechnen der Prüfsumme üblicherweise eine plattformspezifische Funktion (da das Zugreifen auf Inhalte eines Puffers über die Header hinweg üblicherweise eine plattformspezifische Funktion ist). wenn ein BPK **282** ein IP-Paket zu der Plattformumgebung **210** zur Übertragung überleitet, kann es anzeigen, ob es eine Prüfsumme möchte oder nicht, die für das Paket berechnet wird. Ein beispielhafter Wert von 0 kann als Prüfsumme gesendet werden, wenn keine Prüfsumme erforderlich ist. Dies ermöglicht es der Plattformumgebung **210**, zum Zeitpunkt des Empfangens eines BP-Segments, festzulegen, ob eine Prüfsummenverifikation erforderlich ist oder nicht. Dies ist nützlich, wenn man davon ausgeht, dass ein PEP-Endpunkt, insbesondere ein PEP-Endpunkt auf einer Hubseite, BP-Segmente von einigen Backbone-Verbindungen empfangen könnte, deren Prüfsummen ausgeschaltet sind, und von einigen Backbone-Verbindungen, deren Prüfsummen eingeschaltet sind.

[0125] Es gibt viele Gründe, warum eine Backbone-Verbindung ausfallen kann. Allgemein kann eine ausgefallene Backbone-Verbindung über Neuübertragungs-Timeouts erfasst werden. Gründe, weswegen eine Backbone-Verbindung ausfallen könnte, sind:
Eine Verbindung bzw. ein Link zwischen zwei PEP-Endpunkten **705** kann ausfallen. Beispielsweise kann eine Weltraumverbindung aufgrund von Dämpfung durch Regen ausfallen, wenn keine alternativen Verbindungen verfügbar sind oder bei alternativen Verbindungen mit Pfadauswahlregeln, die einige oder alle Backbone-Verbindungen gegen die Verwendung dieser Verbindungen sperren;
Eine Komponente zwischen zwei PEP-Endpunkten **705** kann ausfallen und eine oder alle Verbindungen zwischen den Endpunkten **705** abbrechen lassen. In diesem Fall, wenn nicht alle Verbindungen ausgefallen sind, können Pfadauswahlregeln wiederum einige Backbone-Verbindungen gegen das Tragen alternativer Verbindungen sperren;
Ein PEP-Endpunkt **705** kann ausfallen. Falls ein PEP-Endpunkt **705** ausfällt, fallen üblicherweise alle Backbone-Verbindungen mit diesem aus. Es sei angemerkt, dass, falls ein PEP-Endpunkt **705** wieder startet, der andere PEP-Endpunkt **705** den Fehler erfassen kann, indem ein PBP-<SYN>-Segement für eine Backbone-Verbindung empfangen wird und der Fehler nicht über die Wiederübertragungs-Timeouts erfasst wird.

[0126] Protokollfehler. BPK **282** kann ein ungültiges (nicht im aktuellen Kontext bzw. Umfeld) PBP-Segment

empfangen, das die Backbone-Verbindung in einen Zustand versetzt, der nur durch Neustart der Verbindung verlassen werden kann. Protokollfehler sollten in der Theorie niemals auftreten, da sie wahrscheinlich Implementierungs- oder Entwurfsfehler darstellen. In der Realität können solche Fehler jedoch auftreten. BPK **282** wird im Allgemeinen auf jedes ungültige Segment mit einem PBP-<RST>-Segment antworten, um die Verbindung zum Neustart zu zwingen.

[0127] Der BPK **282** muss keine spezielle Handhabung der Verbindungsprioritäten ausführen. Die BPK **282** kann die Verbindungspriorität als eine Art von „Port"-Nummer behandeln, die es benutzt, um mehrere gleichzeitige Verbindungen zu dem gleichen Ziel-BPK-Partner zu unterstützen. Die Prioritätshandhabung von Verbindungen kann außerhalb des BPK **282** stattfinden, mittels der Größe des Pufferplatzes, auf den BPK **282** für eine bestimmte Verbindung Zugriff hat, und mittels der Geschwindigkeit, mit der BP-Segmente an das WAN über den PK **284** aktuell sendet.

[0128] Ein Pfad-MTU-Erkennungsalgorithmus ist bei TCP-Verbindungen einsetzbar. Der Pfad-MTU-Erkennungsalgorithmus wird in RFC **1191** beschrieben. Da die Nachrichten, auf denen der Algorithmus basiert, ICMP-Nachrichten sind, kann der BPK **282** auch eine Pfad-MTU-Erkennung implementieren. Das Problem, einen Pfad-MTU erkennen zu müssen, kann vermieden werden, indem ein Worst-Case-Wert für den konfigurierten Pfad MTU verwendet wird. Falls jedoch die Verwendung der Pfaderkennung aus Effizienzgründen gewünscht ist, kann der BPK **282** jede empfangene ICMP „Datagram-Too-Big"-Nachricht verwenden, um die maximale Größe der Datensegmente einzustellen, die er sendet.

[0129] Wie zuvor angegeben, kann der BPK **282** unterschiedliche Backbone-Protokolle zur Verwendung mit unterschiedlichen Backbone-Verbindungstypen implementieren. In einigen Fällen wird das implementiere Backbone-Protokoll durch Software festgelegt, die in der Plattform **210** eingebaut ist, basierend auf der Umgebung, in der die Plattform eingebaut werden wird. In anderen Fällen können mehrere Backbone-Protokolle durch BPK **282** implementiert werden, wobei das einzelne Protokoll, das mit einer bestimmten Backbone-Verbindung benutzt wird, von dem Benutzer festgelegt wird, wenn der PEP-Endpunkt konfiguriert wird. Im letztgenannten Fall können die einzelnen Protokolle, die für jede Backbone-Verbindung verwendet werden, in einem PEP-Endpunkt-Profil **701**, **703** spezifiziert werden.

[0130] Fig. 13 zeigt ein Computersystem **1301**, auf dem eine Ausführungsform entsprechend der vorliegenden Erfindung implementiert sein kann. Ein solches Computersystem **1301** kann als Server konfiguriert sein, um Code auszuführen, der die PEP-Funktionen des PEP-Endpunkts **210** ausführt, wie zuvor diskutiert. Das Computersystem **1301** umfasst einen Bus **1303** oder einen anderen Kommunikationsmechanismus zum Kommunizieren von Information, und einen Prozessor **1305**, der mit dem Bus **1305** zur Bearbeitung der Information verbunden ist. Das Computersystem **1301** umfasst ebenfalls einen Hauptspeicher **1307**, wie beispielsweise einen Speicher mit wahlfreiem Zugriff (RAM) oder einer anderen dynamischen Speichervorrichtung, der mit dem Bus **1303** verbunden ist, um Information und Befehle zu speichern, die vom Prozessor **1305** ausgeführt werden sollen. Zusätzlich kann der Hauptspeicher **1307** zum Speichern temporärer Variablen oder anderer Zwischeninformationen während der Ausführung von Instruktionen, die von dem Prozessor **1305** ausgeführt werden sollen, verwendet werden. Der Hauptspeicher **1307** kann ebenfalls zum Speichern der PEP-Steuerungsblöcke verwendet werden, insbesondere mit Bezug auf die vorliegende Erfindung, BCBs, und Puffer, der zum Speichern von Paketen verwendet wird. Das Computersystem **1301** umfasst ferner einen Nur-Lese-Speicher (ROM) **1309** oder eine andere statische Speichervorrichtung, die mit dem Bus **1303** zum Speichern statischer Information und Befehlen für den Prozessor **1305** verbunden ist. Eine Speichervorrichtung **1311**, wie beispielsweise eine magnetische Platte oder eine optische Platte, sind vorgesehen und mit dem Bus **1303** zum Speichern von Informationen und Befehlen verbunden.

[0131] Das Computersystem **1301** kann über einen Bus **1303** mit einem Display **1313** verbunden sein, wie beispielsweise eine Kathodenstrahlröhre (CRT), um Information einem Computerbenutzer darzustellen. Eine Eingabevorrichtung **1315**, einschließlich alphanumerischer und anderer Tasten, ist in dem Bus **1303** zur Kommunikation von Information und Befehlsauswahlen für den Prozessor **1305** verbunden. Ein anderer Typ einer Benutzereingabevorrichtung ist eine Kursorsteuerung **1317**, wie beispielsweise eine Maus, ein Trackball oder Kursor-Richtungstasten zur Kommunikation der Richtungsinformation und zur Befehlsauswahl für den Prozessor **1305** und zum Steuern der Kursorbewegung auf dem Display **1313**.

[0132] Ausführungsformen werden auf die Verwendung eines Computersystems **1301** bezogen, um die PEP-Funktionen des PEP-Endpunkts **210** auszuführen. Entsprechend einer Ausführungsform wird dieser automatische Aktualisierungslösungsweg von einem Computersystem **1301** bereitgestellt abhängig von einem Prozessor **1305**, der ein oder mehrere Sequenzen eines oder mehrerer Befehle ausführt, die in dem Haupt-

speicher **1307** enthalten sind. Solche Befehle können in dem Hauptspeicher **1307** von einem anderen computerlesbaren Medium, wie beispielsweise der Speichervorrichtung **1311**, gelesen werden. Das Ausführen der Befehls-Sequenzen, die in dem Hauptspeicher **1307** enthalten sind, veranlasst den Prozessor **1305**, die Prozessschritte, die hier beschrieben sind, auszuführen. Ein Prozessor oder mehrere Prozessoren in einer Mehrprozessoranordnung können auch verwendet werden, um die Sequenzen der Befehle auszuführen, die in dem Hauptspeicher **1307** enthalten sind. In alternativen Ausführungsformen können festverdrahtete Schaltungen an Stelle oder in Kombination mit Softwarebefehlen verwendet werden. Somit sind die Ausführungsformen nicht auf eine spezifische Kombination einer Hardwareschaltung und Software beschränkt.

[0133] Der Ausdruck „computerlesbares Medium", wie er hier verwendet wird, betrifft jedes Medium, das an einem Bereitstellen von Befehlen für den Prozessor **1305** teilnehmen kann, um die PEP-Funktionen des PEP-Endpunkts **210** auszuführen. Ein solches Medium kann viele Formen annehmen, einschließlich aber nicht begrenzt auf nichtflüchtige Medien, flüchtige Medien und Übertragungsmedien. Nichtflüchtige Medien umfassen beispielsweise optische oder magnetische Platten, wie beispielsweise die Speichervorrichtung **1311**. Flüchtige Medien umfassen dynamische Speicher, wie beispielsweise der Hauptspeicher **1307**. Übertragungsmedien umfassen Koaxialkabel, Kupferdrähte und Lichtleiter, einschließlich der Leitungen, die der Bus **1303** umfasst. Übertragungsmedien können auch die Form akustischer Wellen oder Lichtwellen annehmen, wie beispielsweise jene, die während einer Funk- oder Infrarot-Datenkommunikation erzeugt werden.

[0134] Übliche Formen von computerlesbaren Medien umfassen beispielsweise Floppydisks, flexible Platten, Festplatten, Magnetbänder oder andere magnetische Medien, CD ROM, andere optische Medien, Lochkarten, Papierstreifen, oder andere physikalische Medien mit Lochmustern, RAM, PROM und EPROM, FLASH-EPROM, andere Speicherchips oder Karten, Trägerwellen wie hier beschrieben oder jedes andere Medium, von dem ein Computer lesen kann.

[0135] Verschiedene Formen computerlesbarer Medien können beim Tragen einer oder mehrerer Sequenzen von einem oder mehreren Befehlen an den Prozessor **1305** zur Ausführung beteiligt sein. Beispielsweise können die Befehle zuerst auf einer Magnetplatte eines entfernten Computers liegen. Der entfernte Computer kann die Befehle bezüglich der Ausführung der PEP-Funktionen des PEP-Endpunkts **210** in seinen dynamischen Speicher laden und die Befehle über eine Telefonleitung unter Verwendung eines Modems übersenden. Ein Modem an dem lokalen Computersystem **1301** kann die Daten per Telefonleitung empfangen und einen Infrarotsender verwenden, um die Daten in ein Infrarotsignal umzuwandeln. Ein Infrarotdetektor, der mit dem Bus **1303** verbunden ist, kann die Daten empfangen, die in dem Infrarotsignal getragen werden, und kann die Daten auf den Bus **1303** bringen. Der Bus **1303** trägt die Daten zu dem Hauptspeicher **1307**, von dem aus der Prozessor **1305** die Befehle abruft und ausführt. Die Befehle, die durch den Hauptspeicher **1307** empfangen werden, können optional auf der Speichervorrichtung **1311** gespeichert werden, entweder vor oder nach der Ausführung durch den Prozessor **1305**.

[0136] Das Computersystem **1301** umfasst ebenfalls ein oder mehrere Kommunikationsschnittstellen **1319**, die mit dem Bus **1303** verbunden sind. Die Kommunikationsschnittstellen **1319** stellen eine Zweiwege-Datenkommunikation bereit, die die Netzwerkverbindungen **1321** und **1322** verbindet, die mit einem Local-Area-Network (LAN) **1322** bzw. einem Wide-Area-Network (WAN) **1324** verbunden sind. Das WAN **1324** entsprechend einer Ausführungsform der vorliegenden Erfindung kann ein Satellitennetzwerk sein. Beispielsweise kann die Kommunikationsschnittstelle **1319** eine Netzwerkschnittstellenkarte sein, die in jedes paketvermittelte LAN gehängt werden kann. Ein anderes Beispiel einer Kommunikationsschnittstelle **1319** kann eine asymmetrische digitale Teilnehmerleitung(ADSL)Karte sein, eine Integrated-Services-Digital-Network(ISDN)Karte, ein Kabelmodem oder ein Modem, um eine Datenkommunikationsverbindung mit einem entsprechenden Telefonleitungstyp bereitzustellen. Drahtlose Verbindungen können ebenfalls implementiert werden. In einer solchen Implementierung sendet und empfängt die Kommunikationsschnittstelle **1319** elektrische, elektromagnetische oder optische Signale, die digitale Datenströme tragen, die verschiedene Informationstypen darstellen.

[0137] Netzwerkverbindung **1321** stellt typischerweise eine Datenkommunikation über ein oder mehrere Netzwerke mit anderen Datenvorrichtungen bereit. Beispielsweise kann die Netzwerkverbindung **1321** eine Verbindung durch ein Local-Area-Netzwerk **1323** zu einem Hostcomputer **1325** oder zu einem Datengerät bereitstellen, das von einem Internet-Serviceprovider (ISP) **1327** betrieben wird. ISP **1327** stellt wiederum Datenkommunikationsdienste über das Internet **505** bereit. Zusätzlich wird das LAN **1323** mit einem Intranet **1329** verbunden. Das Intranet **1329**, das LAN **1323** und das Internet **505** verwenden alle elektrische, elektromagnetische oder optische Signale, die digitale Datenströme tragen. Die Signale durch die verschiedenen Netzwerke und die Signale auf der Netzwerksverbindung **1321** und durch die Kommunikationsschnittstelle **1319**, die die digitalen Daten zu und von dem Computersystem **1301** tragen, sind beispielhafte Formen von Trägerwellen,

die Information transportieren.

[0138] Das Computersystem **1301** kann Nachrichten senden und Daten, einschließlich Programmcodes, über das Netzwerk/die Netzwerke, die Netzwerksverbindung **1321** und Kommunikationsschnittstelle **1319** empfangen. Bei dem Internetbeispiel könnte ein Server **1331** einen Anforderungscode für ein Anwendungsprogramm über das Internet **505**, den ISP **1327**, das LAN **1323** und die Kommunikationsschnittstelle **1319** senden.

[0139] Der empfangene Code kann von dem Prozessor **1305** ausgeführt werden, wenn er empfangen wird, und/oder in der Speichervorrichtung **1311** oder einem anderen nichtflüchtigen Speicher für eine spätere Ausführung gespeichert wird. Auf diese Weise kann das Computersystem **1301** Anwendungscode in Form von Trägerwellen erhalten.

[0140] Das Computersystem **1701** kann Meldungen senden und Daten, einschließlich Programmcode, über das Netzwerk/die Netzwerke, Netzwerksverbindung **1721** und Kommunikationsschnittstelle **1719** empfangen.

[0141] Die hier beschriebenen Techniken bieten verschiedene Vorteile gegenüber den bisherigen Lösungen, um die Netzwerkleistung zu verbessern, insbesondere in paketvermittelten Netzwerken wie das Internet. Ein lokaler PEP-Endpunkt und ein entfernter PEP-Endpunkt kommunizieren, um den Austausch von Daten über eine TCP-Spoofing-Funktionalität zu optimieren. Ein Backbone-Protokoll-Kernel liefert eine deutliche Flexibilität und eine auf die Backbone-Verbindung zugeschnittene Leistung in der Unterstützung der PEP-Funktionalität, indem ein Backbone-Protokoll geeignet für eine bestimmte Backbone-Verbindung implementiert wird, und indem unterschiedliche Backbone-Protokolle für unterschiedliche Backbone-Verbindungen unterstützt werden.

[0142] Es ist klar, dass zahlreiche Modifikationen und Variationen der vorliegenden Erfindung im Lichte der vorherigen Lehren möglich sind. Es versteht sich deshalb, dass die Erfindung innerhalb des Rahmens der angehängten Ansprüche auch anders als hier speziell beschrieben ausgeführt werden kann.

Patentansprüche

1. Verfahren zum Routen von Information mit:
Empfangen der Information und Backbone-Verbindungsparametern von einer Plattform (**210**), die ausgelegt ist, um Leistungsverbesserungsfunktionen zu unterstützen, die TCP-Spoofing umfassen, wobei eine Backbone-Verbindungsvorrichtung (**282**) ein Profil aufrecht erhält, das die Backbone-Verbindungsparameter umfasst, die mit einer Backbone-Verbindung zu einer Peer-Plattform (**210**) verknüpft sind, wobei die Backbone-Verbindungsparameter eine Information umfassen, die die Komprimierung des Verkehrs über die Backbone-Verbindung betrifft und Information umfasst, die das TCP-Spoofing betrifft; und
Routen der Information durch die Plattform (**210**) über die Backbone-Verbindung gemäß dem Profil.

2. Verfahren nach Anspruch 1, ferner mit:
Bestimmen einer einer Vielzahl von Backbone-Verbindungen, die die Backbone-Verbindung zum Erreichen der Peer-Plattform (**210**) umfassen.

3. Verfahren nach Anspruch 2, ferner mit:
Bestimmen der Backbone-Verbindung durch Anwenden einer Zuordnungstabelle.

4. Verfahren nach Anspruch 3, wobei die Zuordnungstabelle Segmentzielkennzeichen Backbonesteuerungsblöcken zuordnet.

5. Verfahren nach Anspruch 4, wobei die Backbonesteuerungsblöcke Information speichern, die die Backbone-Verbindung betrifft.

6. Verfahren nach Anspruch 4, wobei die Zuordnungstabelle Zeiger auf die Backbonesteuerungsblöcke speichert.

7. Verfahren nach Anspruch 1, ferner mit:
Empfangen der Backbone-Verbindungsparameter als eine Datenstruktur von der Plattform (**210**).

8. Verfahren nach Anspruch 1, ferner mit:
Empfangen der Backbone-Verbindungsparameter von der Plattform (**210**) beim Start oder wenn die Plattform

(**210**) aktualisierte Backbone-Verbindungsparameter empfängt.

9. Verfahren nach Anspruch 8, wobei das Profil zumindest ein Backbone-Verbindungsprotokoll für zumindest einen Verbindungstyp umfasst.

10. Verfahren nach Anspruch 9, wobei ein erster Verbindungstyp einen ersten Typ von Backbone-Verbindungsprotokoll betreibt, und ein zweiter Typ von Verbindung einen zweiten Typ von Backbone-Verbindungsprotokoll betreibt.

11. Verfahren nach Anspruch 10, wobei der erste Verbindungstyp und der zweite Verbindungstyp gleichzeitig laufen.

12. Kommunikationssystem (**100**) mit:
einer Plattform (**210**), die ausgelegt ist, um Leistungsverbesserungsfunktionen bereitzustellen, die TCP-Spoofing in Verbindung mit einer Peer-Plattform (**210**) umfassen, wobei die Plattform (**210**) ferner ausgelegt ist, um Information und Backbone-Verbindungsparameter zu liefern;
einer Backbone-Verbindungsvorrichtung (**282**), die ausgelegt ist, um mit der Plattform (**210**) zu kommunizieren, wobei die Backbone-Verbindungsvorrichtungen (**282**) ausgelegt ist, um die Information und die Backbone-Verbindungsparameter von der Plattform (**210**) zu empfangen, wobei die Backbone-Verbindungsvorrichtung (**282**) ein Profil hat, das die Backbone-Verbindungsparameter umfasst, die mit einer Backbone-Verbindung zu der Peer-Plattform (**210**) verknüpft sind, wobei die Backbone-Verbindungsparameter Information aufweisen, die die Komprimierung des Verkehrs über das Backbone betrifft, und Information, die das TCP-Spoofing betrifft, wobei das Kommunikationssystem (**100**) konfiguriert ist, um die Information entsprechend dem Profil zu routen.

13. Kommunikationssystem (**100**) nach Anspruch 12, wobei die Backbone-Verbindungsvorrichtung (**282**) konfiguriert ist, um eine einer Vielzahl von Backbone-Verbindungen zu bestimmen, die die Backbone-Verbindung zum Erreichen der Peer-Plattform (**210**) umfassen.

14. Kommunikationssystem (**100**) nach Anspruch 12, wobei die Backbone-Verbindungsvorrichtung (**282**) ausgelegt ist, um die Backbone-Verbindung durch Anwenden einer Zuordnungstabelle zu bestimmen.

15. Kommunikationssystem (**100**) nach Anspruch 12, wobei die Zuordnungstabelle ausgelegt ist, um Segmentzielkennzeichen Backbonesteuerungsblöcken zuzuordnen.

16. Kommunikationssystem (**100**) nach Anspruch 15, wobei die Backbonesteuerungsblöcke konfiguriert sind, um Information zu speichern, die die Backbone-Verbindung betrifft.

17. Kommunikationssystem (**100**) nach Anspruch 15, wobei die Zuordnungstabelle konfiguriert ist, um Zeiger auf die Backbonesteuerungsblöcke zu speichern.

18. Kommunikationssystem (**100**) nach Anspruch 12, wobei die Backbone-Verbindungsvorrichtung (**282**) ausgelegt ist, um die Backbone-Verbindungsparameter als eine Datenstruktur von der Plattform (**210**) zu empfangen.

19. Kommunikationssystem (**100**) nach Anspruch 12, wobei die Backbone-Verbindungsvorrichtung (**282**) konfiguriert ist, um Backbone-Verbindungsparameter von der Plattform (**210**) beim Start zu empfangen, oder wenn die Plattform (**210**) aktualisierte Backbone-Verbindungsparameter empfängt.

20. Kommunikationssystem (**100**) nach Anspruch 19, wobei das Profil zumindest ein Backbone-Verbindungsprotokoll für zumindest einen Verbindungstyp umfasst.

21. Kommunikationssystem (**100**) nach Anspruch 20, wobei ein erster Verbindungstyp ausgelegt ist, um einen ersten Typ von Backbone-Verbindungsprotokoll ablaufen zu lassen, und ein zweiter Verbindungstyp ausgelegt ist, um einen zweiten Typ von Backbone-Verbindungsprotokoll ablaufen zu lassen.

22. Kommunikationssystem (**100**) nach Anspruch 20, wobei der erste Verbindungstyp und der zweite Verbindungstyp ausgelegt sind, um gleichzeitig zu laufen.

23. Backbone-Verbindungsvorrichtung (**282**) zum Routen von Information, mit:

einem Mittel zum Empfangen der Information und Backbone-Verbindungsparametern;
Mittel zum Aufrechterhalten eines Profils, das die Backbone-Verbindungsparameter enthält, die mit einer Backbone-Verbindung zu einer Peer-Plattform (**210**) verknüpft sind, wobei die Backbone-Verbindungsparameter Information aufweisen, die die Komprimierung des Verkehrs über die Backbone-Verbindung betrifft, und Information, die das TCP-Spoofing betrifft; und
Mittel zum Routen der Information über die Backbone-Verbindung entsprechend dem Profil.

24. Backbone-Verbindungsvorrichtung (**282**) nach Anspruch 23, ferner mit:
Mitteln zum Bestimmen einer einer Vielzahl von Backbone-Verbindungen, die die Backbone-Verbindung zum Erreichen der Peer-Plattform (**210**) umfassen.

25. Backbone-Verbindungsvorrichtung (**282**) nach Anspruch 23, wobei das Mittel zum Bestimmen konfiguriert ist, um die Backbone-Verbindung durch Anwenden einer Zuordnungstabelle zu bestimmen.

26. Backbone-Verbindungsvorrichtung (**282**) nach Anspruch 25, wobei die Zuordnungstabelle angeordnet ist, um Segmentzielkennzeichen Backbonesteuerungsblöcken zuzuordnen.

27. Backbone-Verbindungsvorrichtung (**282**) nach Anspruch 26, wobei die Backbonesteuerungsblöcke konfiguriert sind, um Information zu speichern, die die Backbone-Verbindung betrifft.

28. Backbone-Verbindungsvorrichtung (**282**) nach Anspruch 26, wobei die Zuordnungstabelle angeordnet ist, um Zeiger auf die Backbonesteuerungsblöcke zu speichern.

29. Backbone-Verbindungsvorrichtung (**282**) nach Anspruch 23, wobei die Backbone-Verbindungsvorrichtung (**282**) konfiguriert ist, um die Backbone-Verbindungsparameter als Datenstruktur von der Plattform (**210**) zu empfangen.

30. Backbone-Verbindungsvorrichtung (**282**) nach Anspruch 23, wobei die Backbone-Verbindungsvorrichtung (**282**) konfiguriert ist, um die Backbone-Verbindungsparameter von der Plattform (**210**) beim Start zu empfangen, oder wenn die Plattform (**210**) aktualisierte Backbone-Verbindungsparameter empfängt.

31. Backbone-Verbindungsvorrichtung (**282**) nach Anspruch 30, wobei das Profil zumindest ein Backbone-Verbindungsprotokoll für zumindest einen Typ von Verbindung umfasst.

32. Backbone-Verbindungsvorrichtung (**282**) nach Anspruch 31, wobei ein erster Typ von Verbindung ausgelegt ist, um einen ersten Typ von Backbone-Verbindungsprotokoll ablaufen zu lassen, und ein zweiter Typ von Verbindung ausgelegt ist, um einen zweiten Typ von Backbone-Verbindungsprotokoll ablaufen zu lassen.

33. Backbone-Verbindungsvorrichtung (**282**) nach Anspruch 32, wobei der erste Typ von Verbindung und der zweite Typ von Verbindung ausgelegt sind, um gleichzeitig zu laufen.

34. Computerlesbares Medium, das eine oder mehrere Sequenzen eines oder mehrerer Befehle zum Routen von Information trägt, wobei die eine oder die mehreren Sequenzen von einem oder mehreren Befehlen Befehle aufweisen, die, wenn sie von einem oder mehreren Prozessoren ausgeführt werden, den einen oder die mehreren Prozessoren die Schritte ausführen lassen:
Empfangen der Information und Backbone-Verbindungsparametern von einer Plattform (**210**), die ausgelegt ist, um Leistungsverbesserungsfunktionen zu unterstützen, die TCP-Spoofing umfassen, wobei eine Backbone-Verbindungsvorrichtung (**282**) ein Profil aufrecht erhält, das die Backbone-Verbindungsparameter umfasst, die mit einer Backbone-Verbindung zu einer Peer-Plattform (**210**) verknüpft sind, wobei die Backbone-Verbindungsparameter eine Information umfassen, die die Komprimierung des Verkehrs über die Backbone-Verbindung betrifft und Information umfasst, die das TCP-Spoofing betrifft; und
Routen der Information durch die Plattform (**210**) über die Backbone-Verbindung gemäß dem Profil.

35. Computerlesbares Medium nach Anspruch 34, ferner mit:
Bestimmen einer einer Vielzahl von Backbone-Verbindungen, die die Backbone-Verbindung zum Erreichen der Peer-Plattform (**210**) umfassen.

36. Computerlesbares Medium nach Anspruch 34, ferner mit:
Bestimmen der Backbone-Verbindung durch Anwenden einer Zuordnungstabelle.

37. Computerlesbares Medium nach Anspruch 36, wobei die Zuordnungstabelle Segmentzielkennzeichen Backbonesteuerungsblöcken zuordnet.

38. Computerlesbares Medium nach Anspruch 37, wobei die Backbonesteuerungsblöcke Information speichern, die die Backbone-Verbindung betrifft.

39. Computerlesbares Medium nach Anspruch 37, wobei die Zuordnungstabelle Zeiger auf die Backbonesteuerungsblöcke speichert.

40. Computerlesbares Medium nach Anspruch 34, ferner mit:
Empfangen der Backbone-Verbindungsparameter als eine Datenstruktur von der Plattform (**210**).

41. Computerlesbares Medium nach Anspruch 34, ferner mit:
Empfangen der Backbone-Verbindungsparameter von der Plattform (**210**) beim Start, oder wenn die Plattform (**210**) aktualisierte Backbone-Verbindungsparameter empfängt.

42. Computerlesbares Medium nach Anspruch 41, wobei das Profil zumindest ein Backbone-Verbindungsprotokoll für zumindest einen Verbindungstyp umfasst.

43. Computerlesbares Medium nach Anspruch 42, wobei ein erster Verbindungstyp einen ersten Typ von Backbone-Verbindungsprotokoll ablaufen lässt, und ein zweiter Verbindungstyp einen zweiten Typ von Backbone-Verbindungsprotokoll ablaufen lässt.

44. Computerlesbares Medium nach Anspruch 43, wobei der erste Verbindungstyp und der zweite Verbindungstyp gleichzeitig laufen.

Es folgen 15 Blatt Zeichnungen

Anhängende Zeichnungen

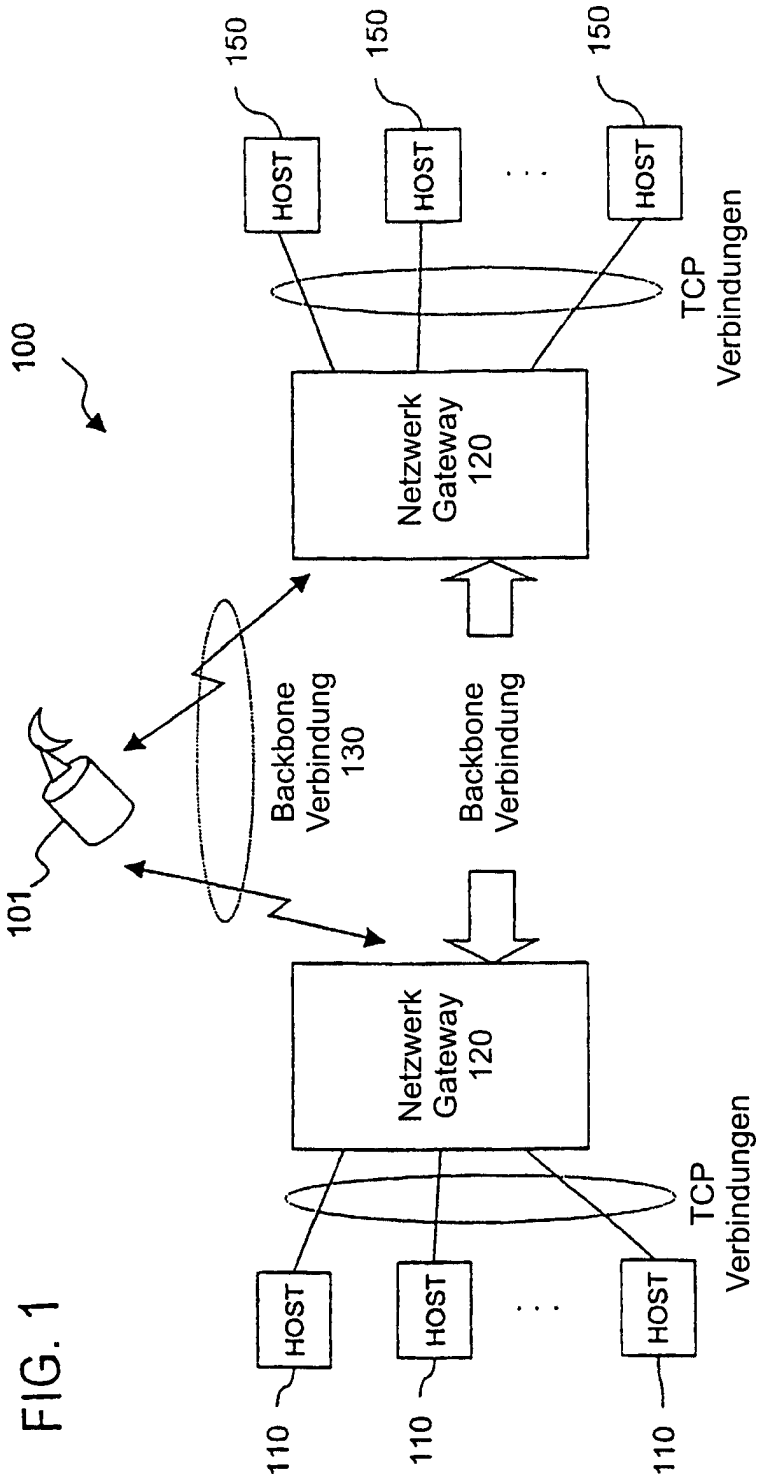


FIG. 1

FIG. 2

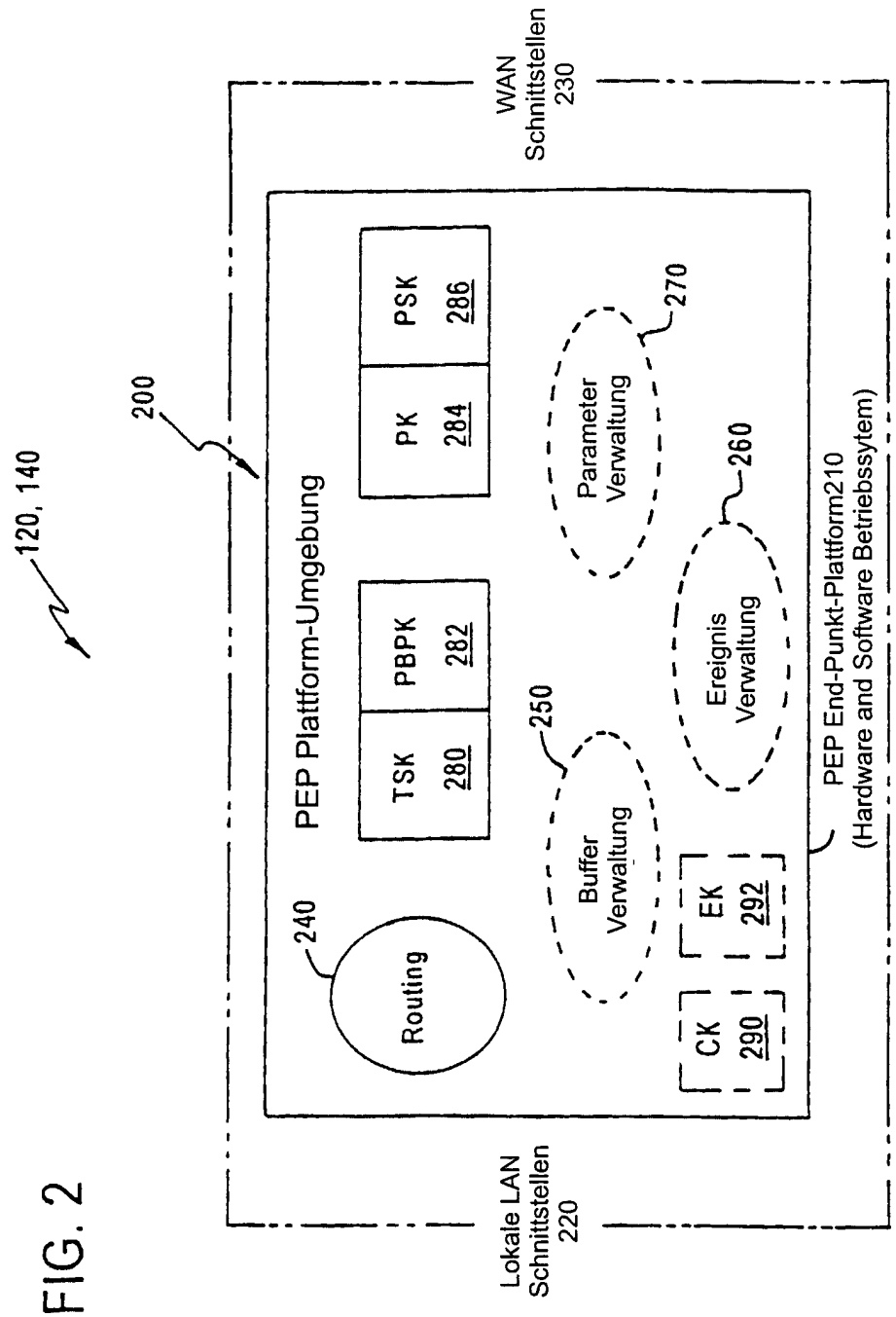

120, 140
200
WAN Schnittstellen 230
PEP Plattform-Umgebung
TSK 280
PBPK 282
PK 284
PSK 286
240
Routing
Buffer Verwaltung
250
Parameter Verwaltung
270
Ereignis Verwaltung
260
CK 290
EK 292
PEP End-Punkt-Plattform210 (Hardware and Software Betriebssytem)
Lokale LAN Schnittstellen 220

FIG. 3

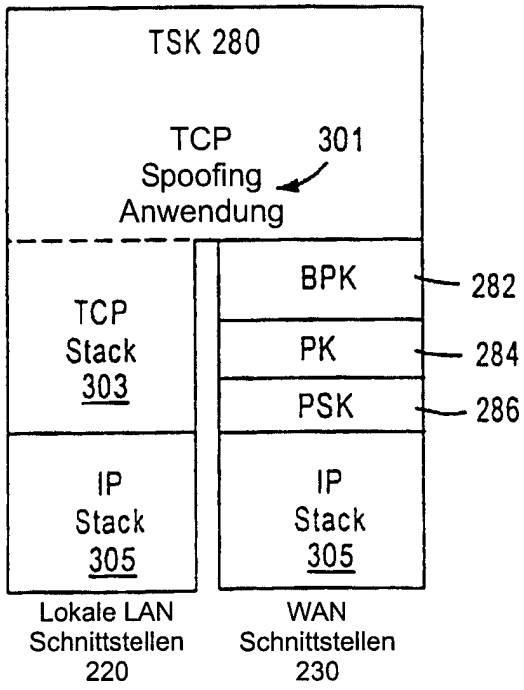

TSK 280
TCP
Spoofing
Anwendung
301
TCP
Stack
303
IP
Stack
305
BPK
282
PK
284
PSK
286
IP
Stack
305
Lokale LAN
Schnittstellen
220
WAN
Schnittstellen
230

FIG. 4A

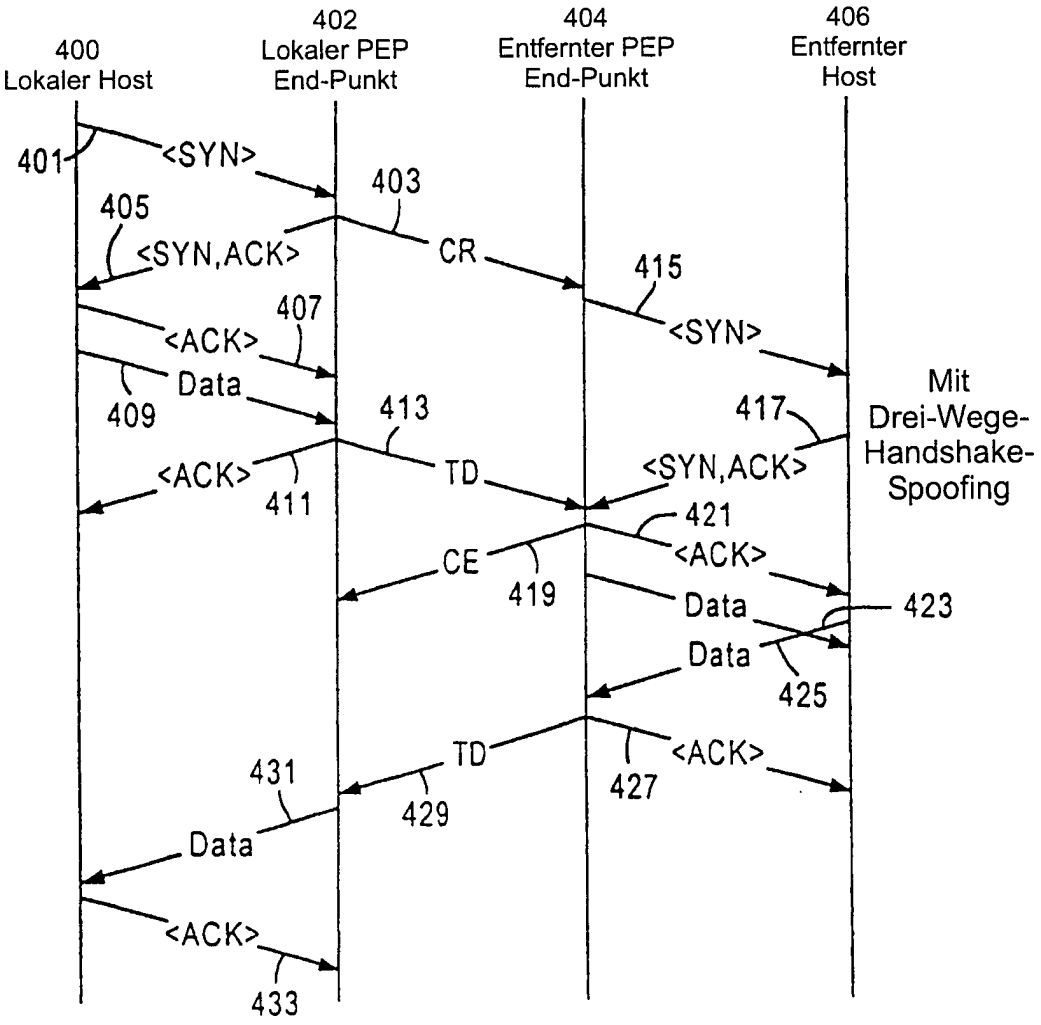

400
Lokaler Host
402
Lokaler PEP
End-Punkt
404
Entfernter PEP
End-Punkt
406
Entfernter
Host
401
<SYN>
403
405
<SYN,ACK>
CR
415
<SYN>
407
<ACK>
Data
409
413
417
Mit
Drei-Wege-
Handshake-
Spoofing
<ACK>
411
TD
<SYN,ACK>
421
<ACK>
CE
419
Data
423
Data
425
TD
<ACK>
431
427
429
Data
<ACK>
433

FIG. 4B

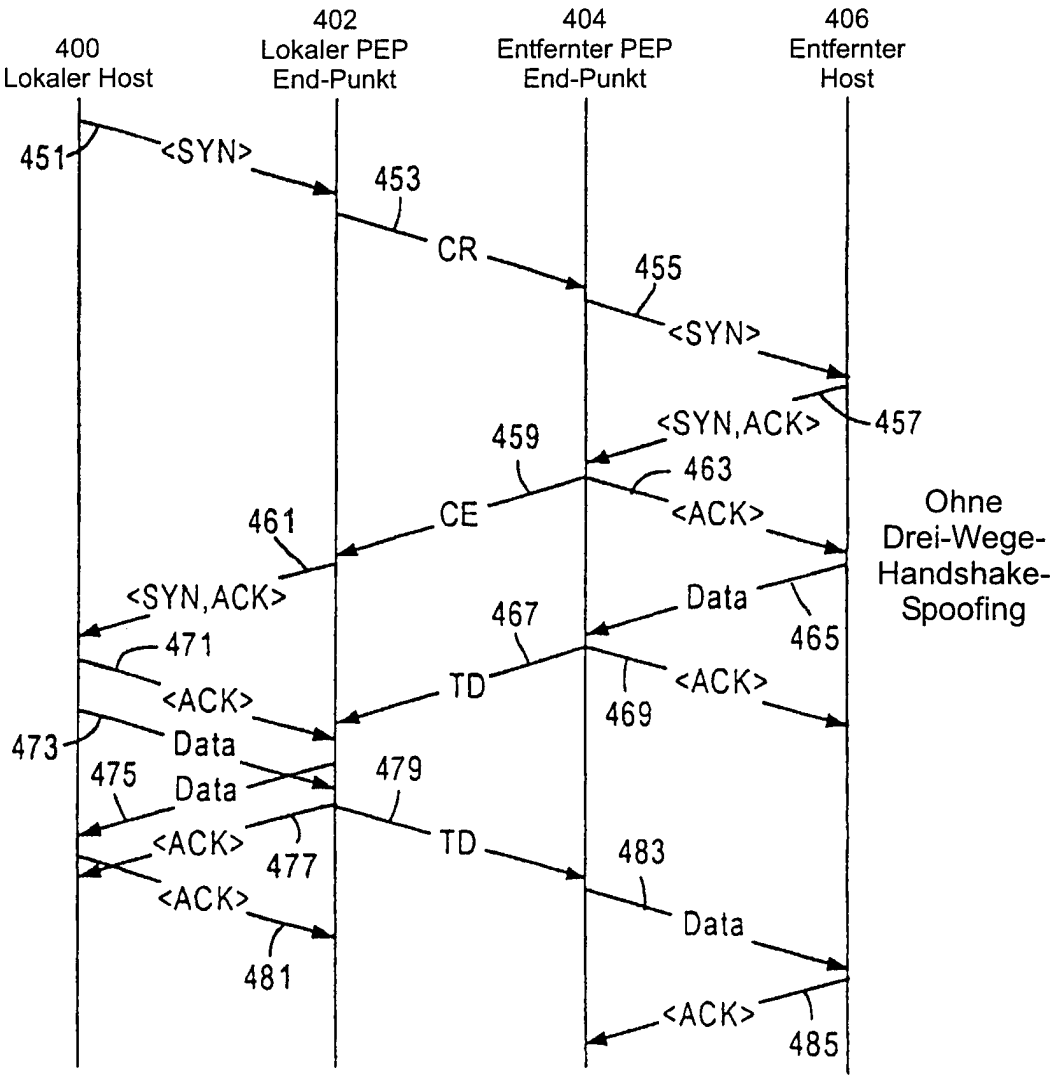
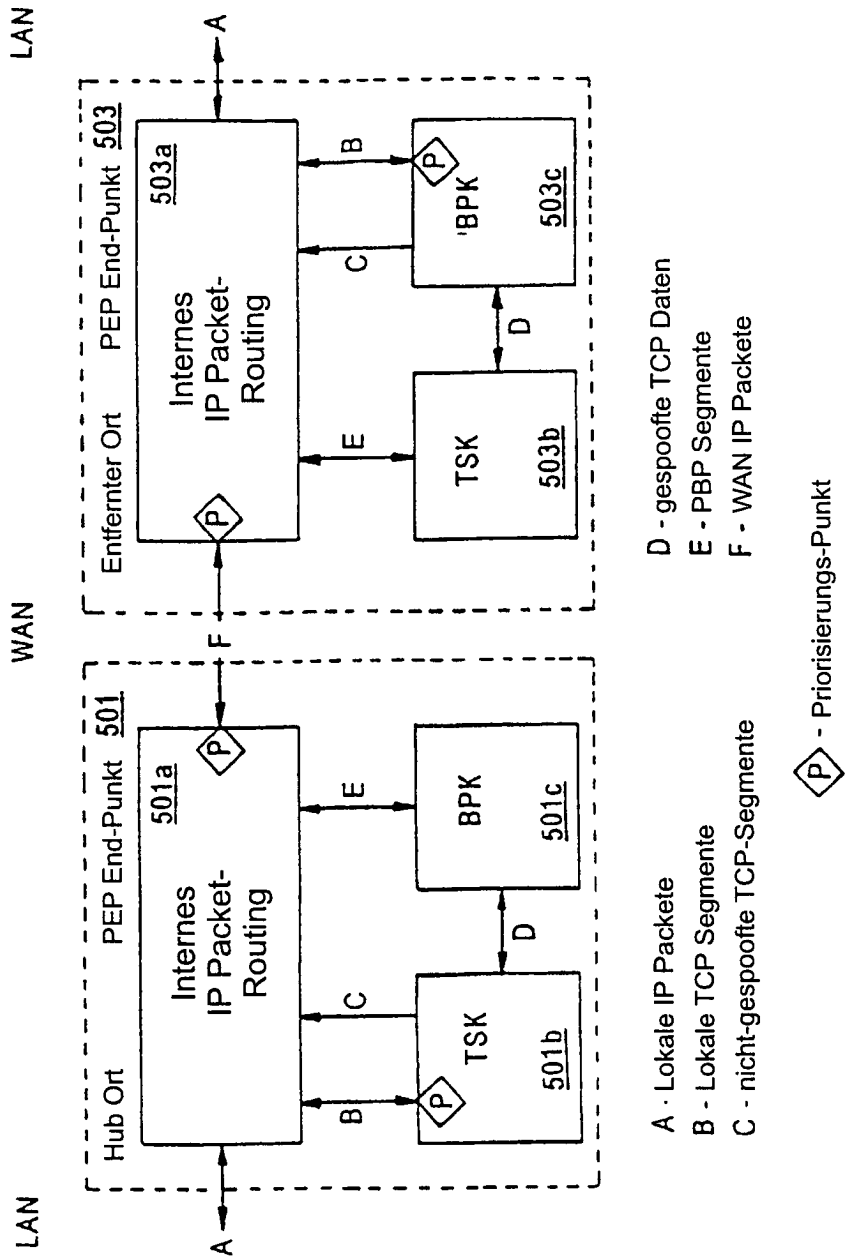

400
Lokaler Host
402
Lokaler PEP
End-Punkt
404
Entfernter PEP
End-Punkt
406
Entfernter
Host
451
<SYN>
453
CR
455
<SYN>
<SYN,ACK>
457
459
463
CE
<ACK>
461
<SYN,ACK>
Data
465
467
471
TD
<ACK>
469
<ACK>
473
Data
475
Data
479
<ACK>
477
TD
483
<ACK>
Data
481
<ACK>
485
Ohne
Drei-Wege-
Handshake-
Spoofing

FIG. 5


LAN
WAN
LAN
Hub Ort
PEP End-Punkt 501
501a
Internes IP Packet-Routing
501b
TSK
501c
BPK
Entfernter Ort
PEP End-Punkt 503
503a
Internes IP Packet-Routing
503b
TSK
503c
BPK
A
B
C
D
E
F
A - Lokale IP Packete
B - Lokale TCP Segmente
C - nicht-gespoofte TCP-Segmente
D - gespoofte TCP Daten
E - PBP Segmente
F - WAN IP Packete
P - Priorisierungs-Punkt

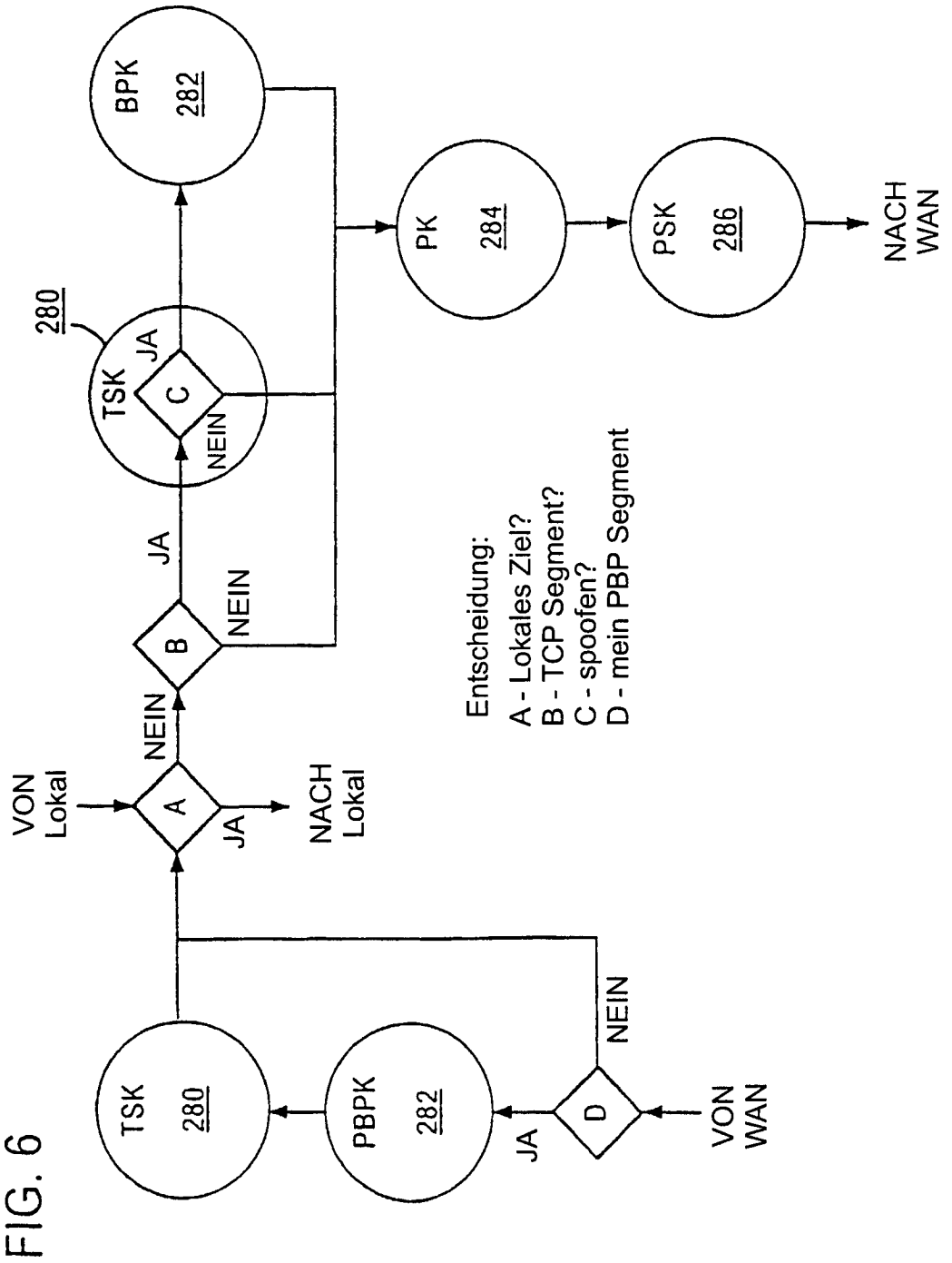

TSK 280
PBPK 282
VON WAN
D
JA
NEIN
VON Lokal
A
NEIN
JA
NACH Lokal
B
JA
NEIN
280
TSK
C
JA
NEIN
BPK 282
PK 284
PSK 286
NACH WAN
Entscheidung:
A - Lokales Ziel?
B - TCP Segment?
C - spoofen?
D - mein PBP Segment

FIG. 6

FIG. 7

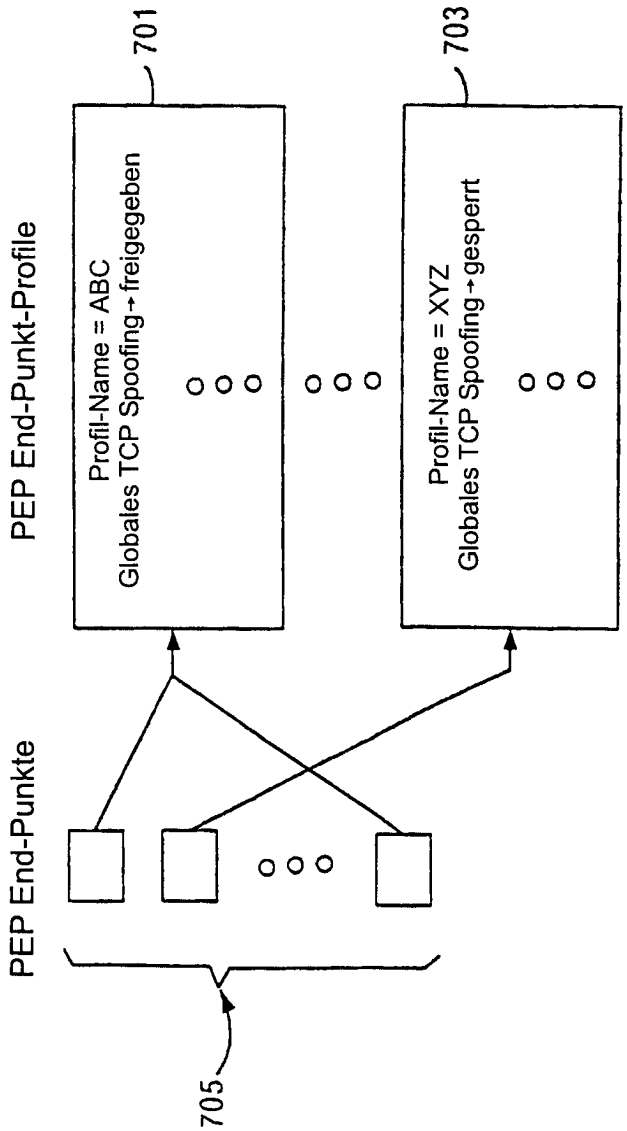

PEP End-Punkte
PEP End-Punkt-Profile
Profil-Name = ABC
Globales TCP Spoofing → freigegeben
Profil-Name = XYZ
Globales TCP Spoofing → gesperrt
701
703
705

FIG. 8

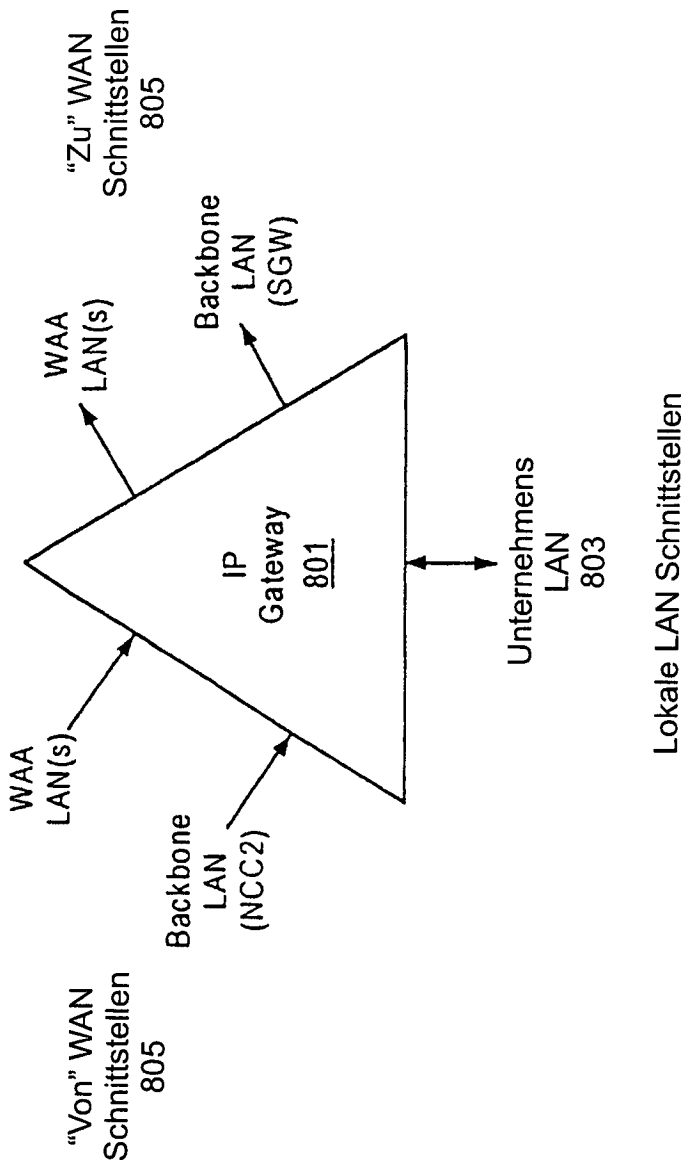

"Zu" WAN Schnittstellen 805
Backbone LAN (SGW)
WAA LAN(s)
IP Gateway 801
Unternehmens LAN 803
Lokale LAN Schnittstellen
WAA LAN(s)
Backbone LAN (NCC2)
"Von" WAN Schnittstellen 805

FIG. 9

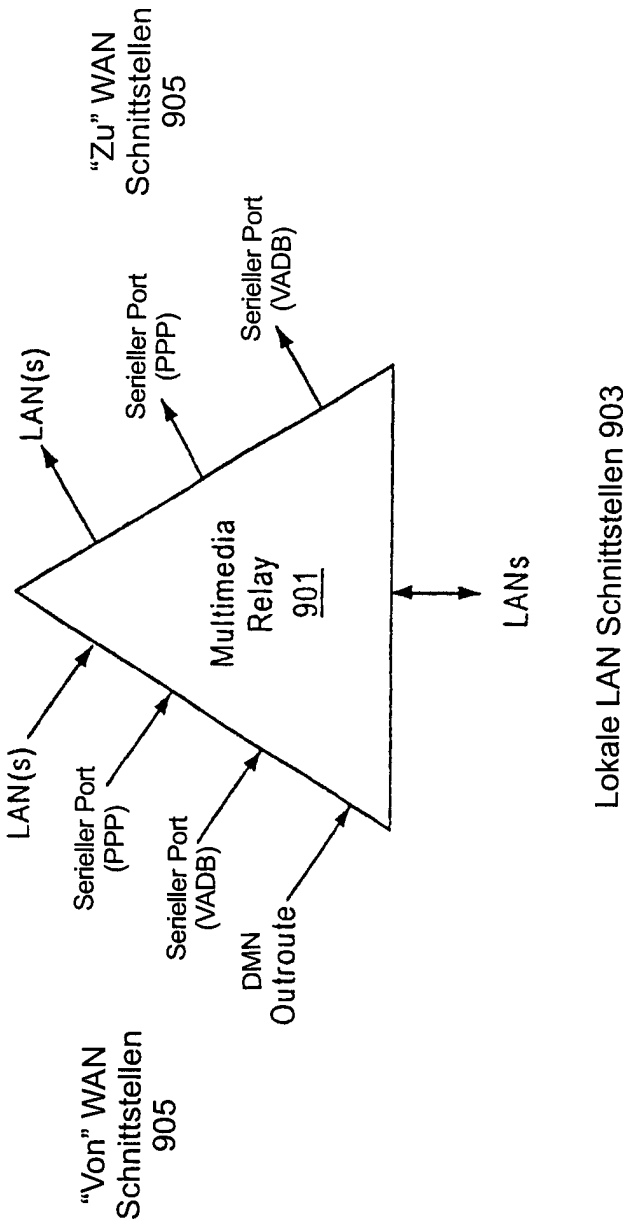

"Zu" WAN Schnittstellen 905
LAN(s)
Serieller Port (PPP)
Serieller Port (VADB)
Multimedia Relay 901
LANs
LAN(s)
Serieller Port (PPP)
Serieller Port (VADB)
DMN Outroute
"Von" WAN Schnittstellen 905
Lokale LAN Schnittstellen 903

FIG. 10

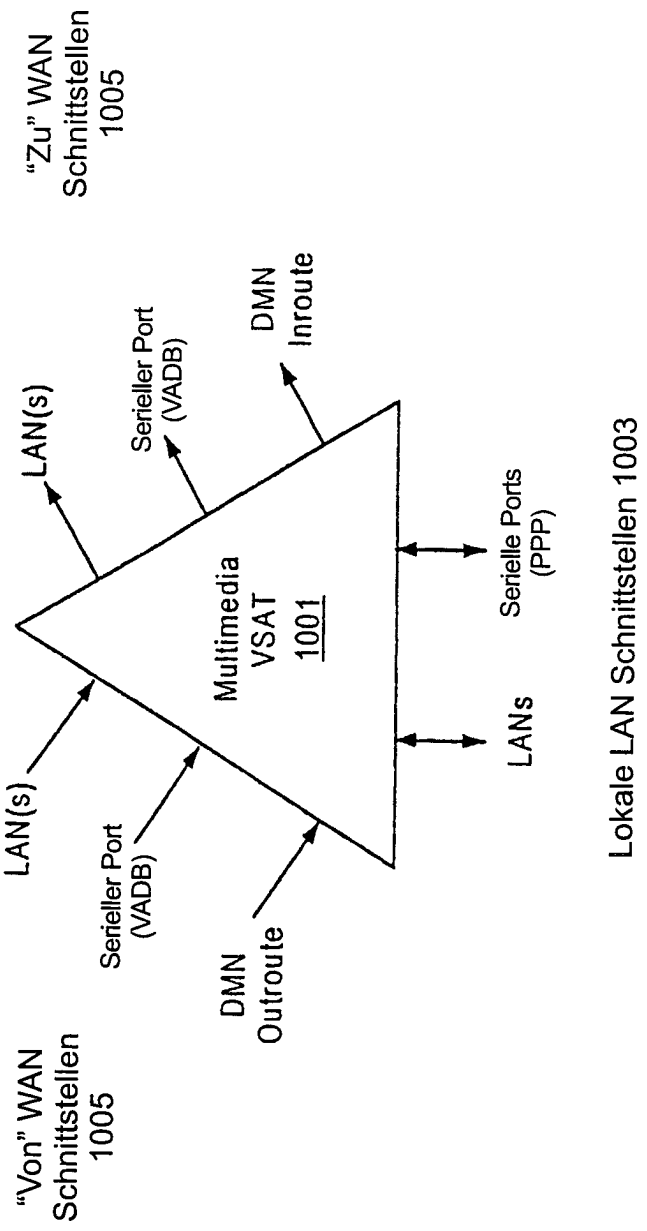
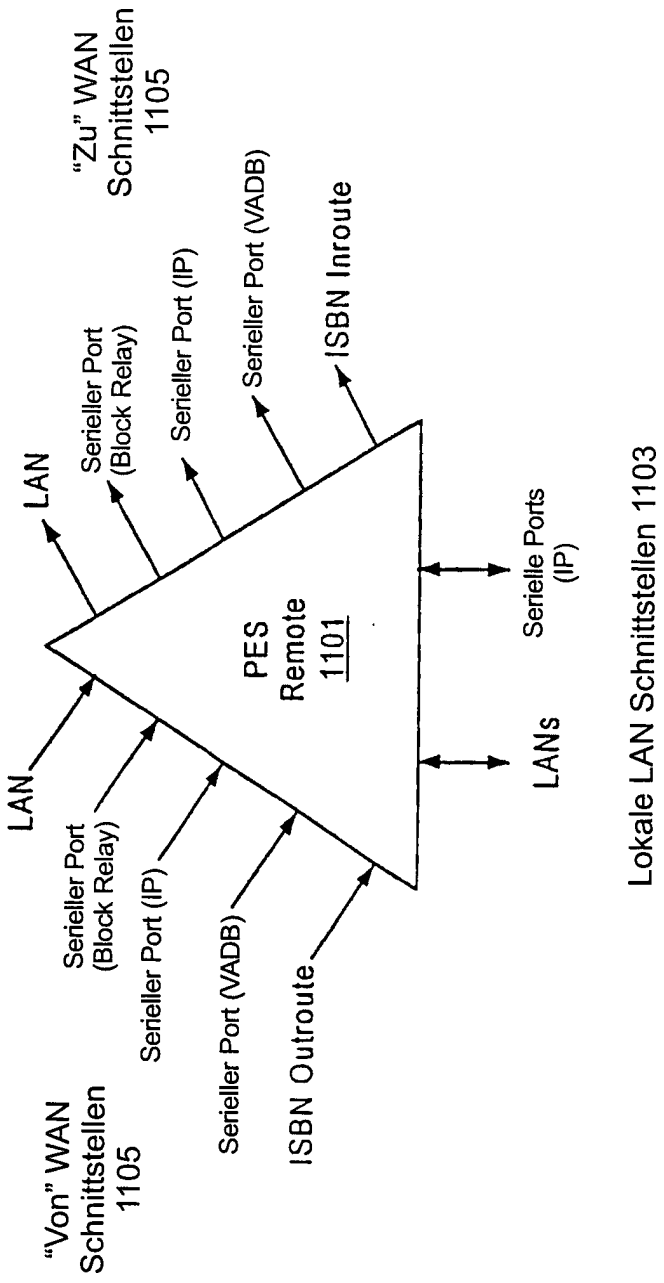

"Zu" WAN Schnittstellen 1005
LAN(s)
Serieller Port (VADB)
DMN Inroute
Multimedia VSAT 1001
Serielle Ports (PPP)
LANs
Lokale LAN Schnittstellen 1003
LAN(s)
Serieller Port (VADB)
DMN Outroute
"Von" WAN Schnittstellen 1005

FIG. 11


"Von" WAN Schnittstellen 1105
LAN
Serieller Port (Block Relay)
Serieller Port (IP)
Serieller Port (VADB)
ISBN Outroute
PES Remote 1101
LANs
Serielle Ports (IP)
Lokale LAN Schnittstellen 1103
"Zu" WAN Schnittstellen 1105
LAN
Serieller Port (Block Relay)
Serieller Port (IP)
Serieller Port (VADB)
ISBN Inroute

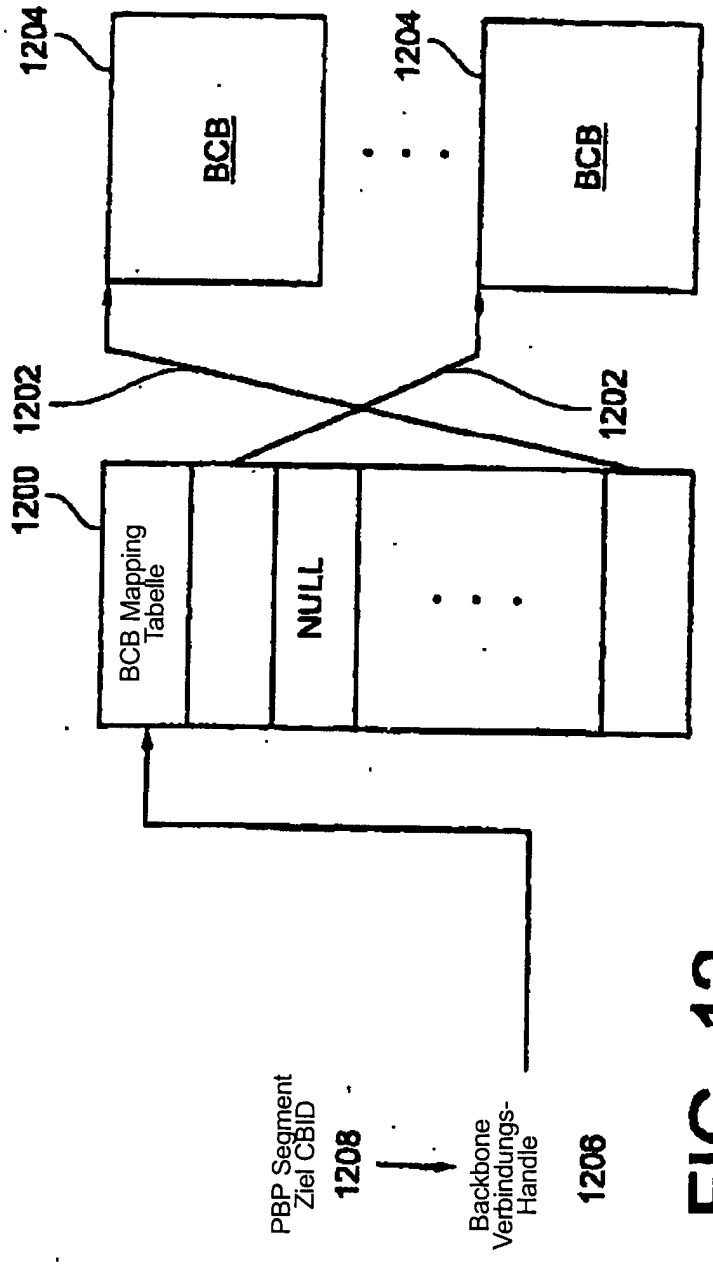

1204
BCB
1204
BCB
1202
1202
1200
BCB Mapping Tabelle
NULL
PBP Segment Ziel CBID
1208
Backbone Verbindungs-Handle
1206

FIG. 12

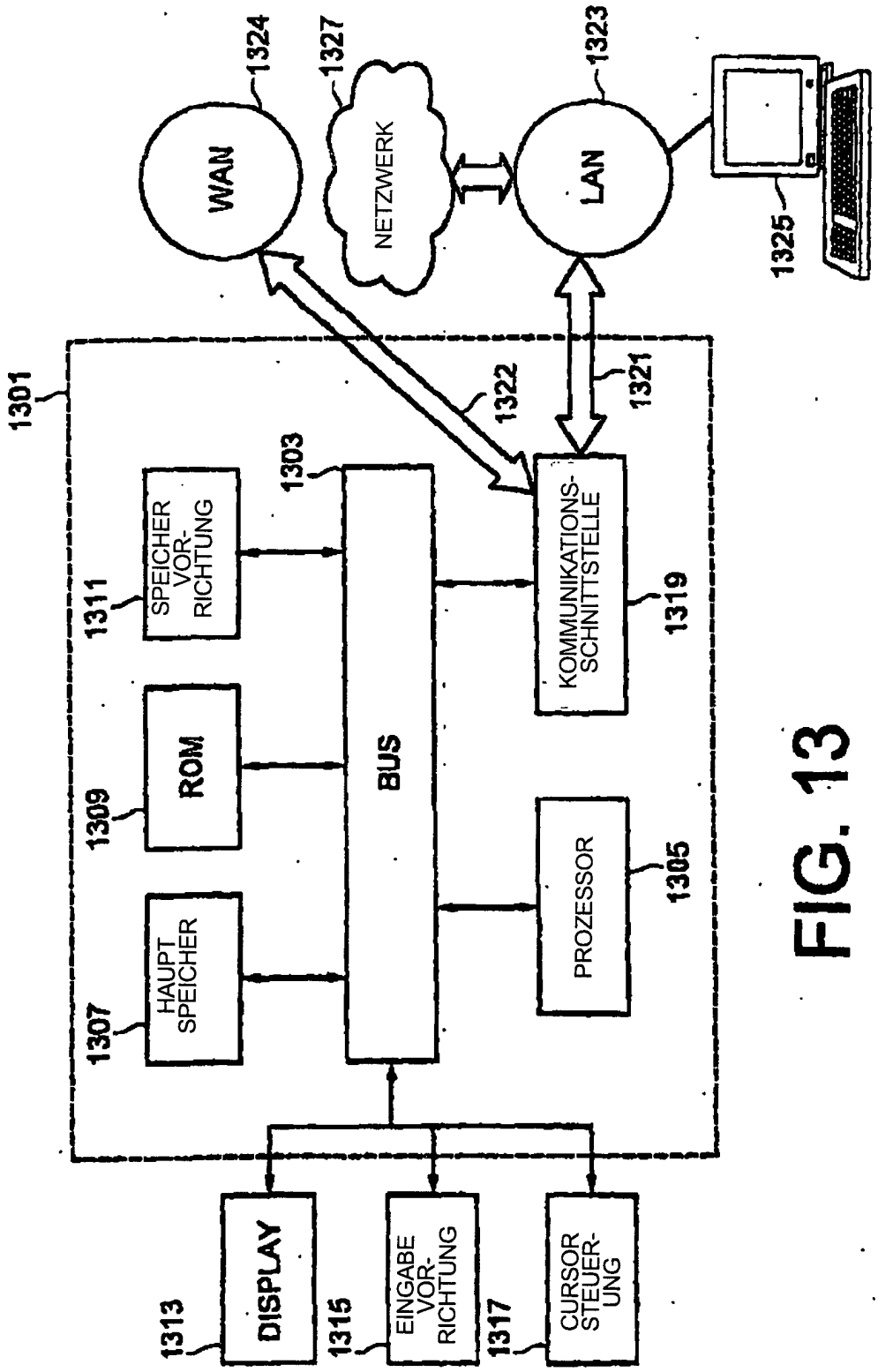

1324
WAN
1327
NETZWERK
1323
LAN
1325
1301
1322
1321
1303
SPEICHER VOR-RICHTUNG
1311
KOMMUNIKATIONS-SCHNITTSTELLE
1319
ROM
1309
BUS
PROZESSOR
1305
HAUPT SPEICHER
1307
DISPLAY
1313
EINGABE VOR-RICHTUNG
1315
CURSOR STEUER-UNG
1317

FIG. 13

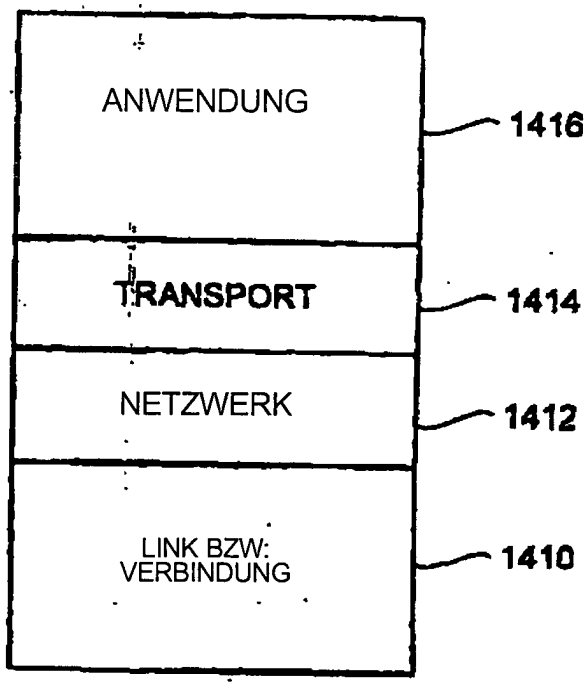
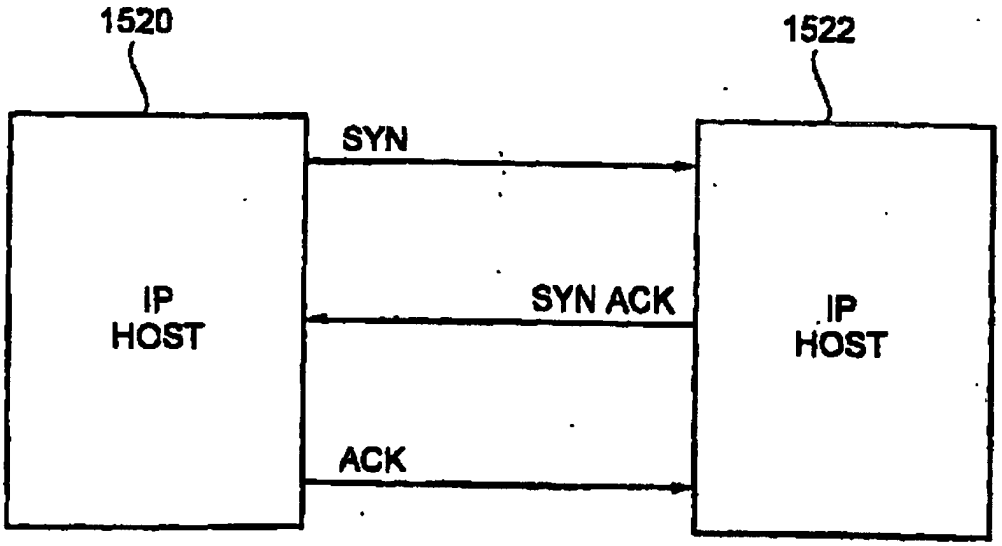


FIG. 14



TCP VERBINDUNG

FIG. 15