PathMTU Discovery mit IP-Optionen

Leopold Franzens Universität Innsbruck
Institut für Informatik

Bakkalaureatsarbeit

Eingereicht bei:
Dr. Dipl. Ing. Michael Welzl

Eingereicht von:
Richard Spindler, 0116225
csad2715@uibk.ac.at

25. Juli 2005

Inhaltsverzeichnis

1 Einleitung

IP, das Internet Protokoll, funktioniert als so genanntes “packet switched” Netzwerk, das heißt Informationen werden in Form von kleinen, voneinander unabhängigen Paketen verschickt. Diese Pakete haben, je nachdem über welches physikalische Netzwerk sie transportiert werden eine bestimmte maximale Größe, die nicht überschritten werden darf. Diese Größe ist allerdings nicht einheitlich, und kann abhängig von der zugrunde liegenden Technik variieren. Gigabit Ethernet unterstützt z.B. wesentlich größere Pakete als ein normales Modem.

Da in einem Netzwerk wie dem Internet unterschiedlichste Geräte gleichzeitig verbunden sind, kann es durchaus vorkommen, dass ein Paket das mehrere Knoten auf dem Weg zum Ziel passiert über Verbindungen mit unterschiedlichen Eigenschaften weitergeleitet wird. Diesen Weg durch das Netzwerk nennt man Pfad. Die Größe eines Pakets das eine einzelne Verbindung passieren kann, wird mit Maximum Transmission Unit kurz MTU bezeichnet, wodurch sich der Begriff PathMTU für das größtmögliche Paket entlang eines Pfades ergibt.

In dem Beispiel in Abbildung 1 wäre die PathMTU zwischen Rechner 1 und Rechner 2 genau 1000, also das Minimum der Verbindungen dazwischen. Wenn Rechner 1 jetzt also ein Paket an Rechner 2 schickt, sollte er darauf achten, dass er die Größe auf 1000 beschränkt, obwohl er über eine Verbindung mit einer MTU von 1500 angebunden ist. Allerdings weiß er das zu diesem Zeitpunkt noch nicht. Hier kommt der Mechanismus der so genannten PathMTU Discovery ins Spiel. Sobald der Router ein zu großes Paket von Rechner 1 erhält, verwirft er es und schickt eine Fehlermeldung an Rechner 1 zurück. Dieser schickt daraufhin nur noch kleinere Pakete. Das ist sozusagen die klassische Variante dieses Verfahrens. Ein kleiner Haken fällt hierbei sofort auf, ein Paket wird mindestens verworfen, falls die Fehlermeldung etwas später kommt vielleicht sogar mehrere.

Um diesen Paketverlust von vornherein zu vermeiden, existiert der Vorschlag, mit Hilfe von so genannten IP-Optionen die MTU entlang des Pfades schon im Vorfeld festzustellen. Die Idee funktioniert folgendermaßen: Rechner 1 schickt ein spezielles Paket, das die MTU der eigenen Verbindung enthält. Passiert das Paket einen Link mit kleinerer MTU, wird dieser Wert herunter gesetzt, und am Ende von Rechner 2 zurückgeschickt. Sobald Rechner 1 die Antwort erhält, wird er nur noch Pakete der passenden Größe schicken, ohne dabei durch eine zu geringe Paketgröße den Link nicht optimal auszunutzen.

2 Verfahren

2.1 IP

Das Internet Protokoll [4] bildet die Grundlage des weltweiten Internet. Es stellt im Wesentlichen einen Mechanismus bereit, um Pakete zwischen Computern im Netzwerk zu verschicken. Dabei besteht ein IP-Paket hauptsächlich aus einer Adresse und einem Daten-Teil. Das Protokoll verfügt über keine Highlevel-Funktionalitäten wie Daten-Integrität oder Verbindungen. Die einzige Prüfsumme die der IP-Header enthält ist die

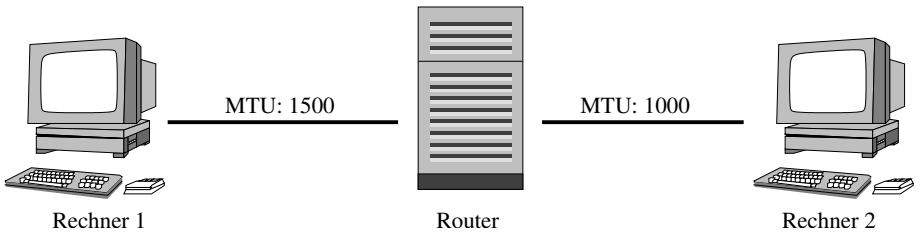

Abbildung 1: PathMTU

für den Header selbst, die Anwendungsdaten müssen in einer darüber liegenden Schicht geprüft werden. Das hat zur Folge, dass man sobald etwas am Header geändert wird, z.B. der Wert einer IP-Option aktualisiert wird, die Prüfsumme ebenfalls neu berechnen muss. Ansonsten enthält ein IP-Paket natürlich Quell- und Zieladdresse, einige Flags für verschiedene Dinge, und wie bereits erwähnt, wahlweise keine, eine oder mehrere IP-Optionen.

Eine wichtige Eigenschaft dieses Protokolls ist, dass es unabhängig von der darunter liegenden Hardware-Konnektivität ist. Das heißt, es lässt sich über verschiedene Medien, sei es nun eine Modem-Verbindung oder ein 100MBit Ethernet verwenden. Allerdings haben unterschiedliche Medien unterschiedliche Eigenschaften, und sind z.B. beschränkt was die maximale Größe eines übertragbaren Pakets betrifft. Das ist insofern kein Problem, als das IP natürlich unterschiedliche Paket-Größen erlaubt, und auch die Möglichkeit bietet Pakete zu fragmentieren, also in mehrere kleinere Pakete zu zerlegen. Dieses Vorgehen ist allerdings nicht immer erwünscht, und daraus ergibt sich die in der Einleitung angesprochene Problematik.

2.1.1 PathMTU Discovery (PMTUD)

Wie bereits erwähnt existiert bereits ein Verfahren das ein Erkennen der PathMTU ermöglicht, es ist im Detail in [5] beschreiben. Um den PMTU Discovery Vorgang anzustoßen wird das so genannte “Don’t Fragment” Bit im IP Paket gesetzt, das besagt, dass zu große Pakete nicht zerkleinert, sondern verworfen werden. Natürlich nicht ohne dabei eine entsprechende Fehlermeldung an den Versender zu schicken. Dabei handelt es sich um eine so genannte “ICMP Destination unreachable” Meldung.

Auf dieser Grundlage funktioniert der PMTU Discovery Algorithmus. Betrachten wir PMTUD in Zusammenhang mit TCP: TCP schickt Segmente mit der Größe der ausgehenden MTU, was natürlich nur dann tatsächlich der Fall ist, wenn entsprechend viele Daten in der Warteschlange stehen. Sollte dabei die tatsächliche PMTU überschritten werden, erhält man die entsprechende ICMP Nachricht zurück. Im Falle der Benachrichtigung ist die maximale Paketgröße bekannt, und kann nun verwendet werden. Nun bleibt die Paket-Größe für eine Weile konstant. Da sich im Laufe der Zeit ein Pfad im IP Netz ändern kann, wird hin und wieder probiert ein größeres Paket zu verschicken, um in einem solchen Fall eine effizientere Auslastung der Verbindung zu erreichen. Die verlorenen Pakete sind insofern kein Problem, als TCP Datenintegrität

gewährleistet, und selbstständig verlorene Pakete nochmal schickt. Der Algorithmus ist also auf die korrekte und zuverlässige Zustellung von ICMP Nachrichten angewiesen, was allerdings auch zum Problem werden kann.

Geht man vom Standpunkt einer strikten Schichten-Trennung innerhalb des Netzwerkmodells aus, so wird klar, dass PMTUD hier einige Querschläge vornimmt. Das Verfahren wird von der verbindungsorientierten TCP Schicht aus initiiert, da es nur Sinn macht, wenn man vor hat einen Pfad über einen längeren Zeitraum hinweg zu benutzen. Die ICMP Nachricht ist allerdings ein Element aus der IP Schicht, da die Benachrichtigung über ein verlorenes Paket unabhängig von der drüber liegenden Transportschicht ist und das Ergebnis, das ja über den IP Layer wieder an den Host zurück geht, muss anschließend wieder an die TCP Schicht übermittelt werden, die dann danach die maximale Größe der erzeugten Segmente richtet.

2.1.2 Die Zukunft von PMTUD

Abgesehen vom bereits existierenden PMTUD, und der hier vorgestellten Variante gibt es noch einen anderen Vorschlag, der in [9] erläutert wird. Das Ziel dieses Verfahrens ist es, die Probleme klassischer PMTUD zu beseitigen, und gleichzeitig kompatibel mit vorhandener Infrastruktur zu bleiben. Dabei ist ist das Verfahren vom Prinzip her ähnlich, da es ebenfalls auf dem "Don't fragment" Bit aufbaut. Allerdings ist es so ausgelegt, das es auch ohne das ICMP basierte Feedback zu einem brauchbaren Ergebnis führt.

Der Vorgang verläuft so, dass der Sender mit kleinen Paketen anfängt, und dann sukzessive für je ein Paket die Paketgröße erhöht, wartet ob dieses ankommt, und falls ja, für das nächste Paket die Größe erhöht. Das wird solange durchgeführt, bis ein Paket verloren geht oder die MTU des ausgehenden Links erreicht ist.

Allerdings zieht auch TCP aus Paketverlust seine Schlüsse und nimmt an, dass Stau im Netzwerk aufgetreten ist. Das zu koordinieren bedarf einigen Aufwands in der Implementierung, damit gewährleistet ist, dass TCP den Verlust eines PMTUD Test Pakets eben nicht als Stau-Signal interpretiert. Deshalb wird es außerdem nötig, dass das Verfahren für jeden Transport Layer eigens implementiert wird. Man spricht dann von PLPMTUD, d.h. Packetization Layer PMTUD.

2.1.3 IP-Optionen

Dieses Verfahren zur Optimierung der PMTUD macht wie bereits erwähnt Gebrauch von so genannten IP-Optionen. Ein IP-Paket enthält optional eine oder mehrere IP-Optionen. Das sind zusätzliche Informationen im IP-Header, die für unterschiedlichste Zwecke genutzt werden können. Der schematische Aufbau eines IP-Pakets ist in Abbildung 2 dargestellt.

Ob ein Paket Optionen enthält, sieht man bereits am IHL (Internet Header Length) Feld, das die Länge des Headers angibt. Sobald es größer als 5 ist, besteht die Möglichkeit nach der "Destination Address" weitere Informationen hinzuzufügen. IP-Optionen stehen im IP-Header an letzter Stelle, und es können nahezu beliebig viele angegeben werden. Da der Verwendungszweck nicht vorhersehbar ist, ist natürlich auch die Länge einer einzelnen Option flexibel. In Abbildung 3 ist dies im Detail ersichtlich.

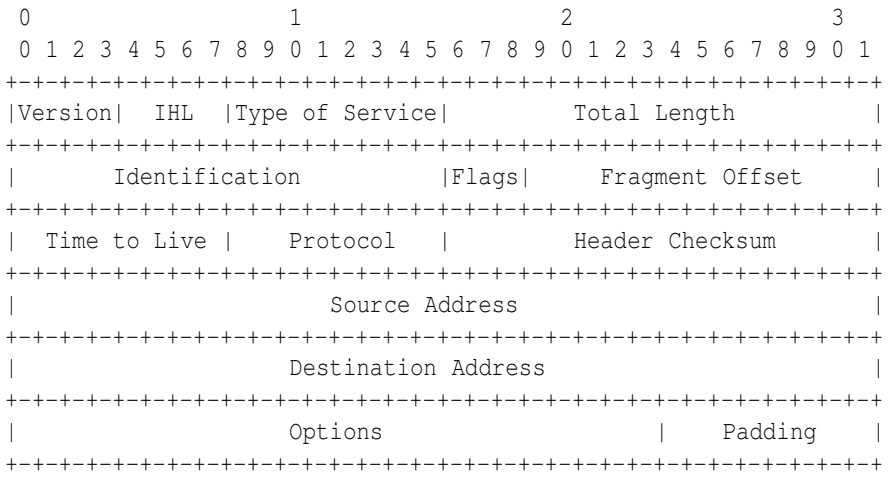

Abbildung 2: IP-Paket

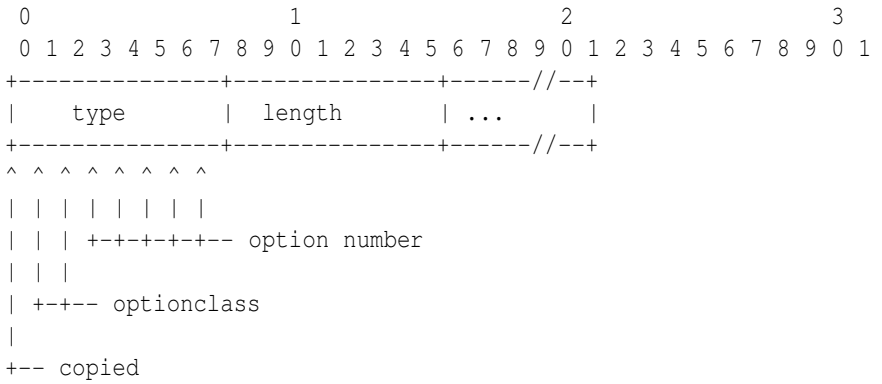

Abbildung 3: IP-Option

Jede Option hat einen Typ, der in [4] festgelegt ist. Je nachdem welcher Typ das ist, gibt es zusätzlich ein Feld, dass die Länge der Option angibt, und danach folgt der Daten-Teil, der den zur Option passenden Inhalt hat. Bis auf die beiden ersten IP-Optionen "End of Option List" und "No Operation" wird stets die Länge angegeben; das ist hilfreich damit ein Gerät, das keine Unterstützung für eine neuere IP-Option hat, diese ignorieren kann. Es wird dann einfach die unbekannte Option übersprungen, und die nächste behandelt.

2.2 PMTU-Optionen

PMTU-Optionen sind nicht als Ersatz für das verwendete PMTUD Verfahren gedacht - allerdings gab es bereits einen Vorschlag, der ausschließlich auf IP-Optionen setzt. Dieser wird in [6] erläutert. Zu diesem Zeitpunkt war jedoch das PMTUD Verfahren noch nicht entwickelt. Der Unterschied zu diesem Vorschlag ist daher, dass hier IP-Optionen als Ergänzung zum existierenden Verfahren verwendet werden. Dadurch wird zum einen gewährleistet, dass auch dann PMTUD funktioniert, wenn die Option von der Gegenstelle, bzw. den beteiligten Routern nicht unterstützt wird. Zum anderen kann man aber Grenzen und Fehlerquellen des Verfahrens beseitigen, bzw. umgehen.

Um PMTUD mit Hilfe von Optionen durchzuführen braucht man zwei verschiedene Typen von IP-Optionen: die Test-Option, die wenn nötig von einem Router unterwegs aktualisiert wird, und die Antwort-Option, die nach erfolgreicher Zustellung dem Absender die ermittelte PMTU schickt.

Während der PMTU Test natürlich auf IP Ebene stattfinden muss, ist dies für die Antwort nicht notwendig, bzw. wird davon sogar abgeraten, da sich durch den erhöhten Aufwand beim Routen die Antwort verzögern könnte. Für diese Bakkalaureatsarbeit wurde der Weg gewählt, die Antwort ebenfalls als IP-Option zu senden, da dadurch die Implementierung einfacher ist.

Nach Abschluss der Test-Phase müssen die ermittelte MTU noch interpretiert und angewendet werden. Diesen Wert daher sofort als optimale PMTU zu behandeln, und ab sofort mit dieser Paketgröße zu senden ist dabei nicht die bevorzugte Strategie. Da anzunehmen ist, dass nicht alle Knoten in einem Netzwerk mit diesem Verfahren kompatibel sind, wird empfohlen, dass man weiterhin auf empfangene ICMP-Nachrichten reagiert, und dann eine kleinere PMTU annimmt, obwohl die IP-Option eventuell einen größeren Wert ermittelt hat. Der bestehende PMTUD Mechanismus bleibt somit aufrecht, und letztendlich die bestimmende Instanz, im Falle dass die ermittelten PMTU Werte voneinander abweichen. Dieser Fall ist recht einfach zu interpretieren: der Router am Bottleneck[1] hat keine Unterstützung für PMTU-Optionen implementiert, und stattdessen auf das ebenfalls weiterhin gesetzte "Don't Fragment" Bit reagiert, das Paket verworfen und eine ICMP Nachricht generiert.

[1] Verbindung mit der kleinsten MTU

3 Motivation

3.1 Probleme mit PMTUD

3.1.1 Black Hole Detection

Black Hole Detection passiert, wenn aus scheinbar unerklärlichen Gründen Pakete verschwinden, sie also in ein "Schwarzes Loch" gesaugt werden. Die Umstände unter denen das Problem auftritt, stehen in direktem Zusammenhang mit dem PMTUD Prozess. Normalerweise zerteilt ein Router Pakete die für ihn zu groß sind in kleinere Pakete, und schickt diese weiter. Da das aber wie schon erwähnt unerwünscht ist, und deswegen das "Don't Fragment" Bit gesetzt ist, darf der Router das Paket nicht zerteilen, und unterlässt dies auch. Stattdessen verwirft er es, generiert die ebenfalls erwähnte ICMP Nachricht, und schickt sie zurück. Damit funktioniert der erste Teil wie geplant. Wenn diese Nachricht den Sender aber nie erreicht, wird der weiter Pakete schicken, in der Annahme, dass alles in Ordnung ist. Da allerdings die Bestätigungspakete der Gegenstelle ausbleiben, wird über kurz oder lang die Verbindung kollabieren, die Pakete gehen auf mysteriöse Weise verloren.

Es gibt zumeist ganz spezielle Gründe warum das passiert, zum Beispiel wenn fehlkonfigurierte Firewalls solche ICMP Nachrichten aus so genannten "Sicherheitsgründen" verwerfen, anstatt sie weiterzuleiten. Das Problem tritt auf, wenn sich eine derartige Firewall zwischen zwei Knoten befindet, von denen einer PMTUD ausführt. Das ICMP Paket wird zwar hinter der Firewall korrekt generiert und verschickt, aber auf dem Rückweg zum Sender verworfen.

Es muss aber nicht unbedingt eine Firewall sein: genauso gut kann passieren, dass das ICMP Paket vom Router gar nicht erst erzeugt wird, weil dieser z.B. entsprechend konfiguriert, bzw. fehlkonfiguriert ist. Die Auswirkungen sind dann allerdings dieselben. Da kleinere Pakete stets durchkommen, funktioniert die Verbindungsaufnahme meist problemlos, erst beim Übertragen größerer Datenmengen, für die möglichst große Pakete verwendet werden, kommt es zu diesem Verhalten, woraus sich auch die grundsätzliche Schwierigkeit der Diagnose ergibt, da mit einem Ping auf den Gegenstellen-Rechner aufgrund der kleineren Paketgröße keinerlei Unregelmäßigkeiten feststellt werden können.

In [7], das all diese Hindernisse anhand von Beispielen vertieft, werden noch einige weitere Probleme mit PMTUD erläutert; diese beziehen sich allerdings nicht ausschließlich auf das Verfahren selbst, sondern auch auf vorhandene Implementierungen, die ungünstige Annahmen treffen.

3.1.2 Bestimmen der MSS mittels der PMTU

MSS bedeutet Maximum Segment Size und ist ein Begriff, der eine Eigenschaft in TCP beschreibt; Sie ist genauer gesagt das Äquivalent zu dem was die MTU für IP ist, für TCP. Im Laufe einer TCP Verbindung wird diese Größe auch an die Gegenstelle übermittelt, sodass diese weiß, welche Größe eines TCP-Pakets der Empfänger im besten Fall verarbeiten kann. Grundsätzlich sollte das der MTU bzw. der PathMTU entsprechen, der Wert kann aber auch kleiner sein, wenn z.B. der Empfänger aus was für Gründen auch immer nur etwas kleinere TCP-Pakete verarbeiten kann oder will.

Daraus ergibt sich auch die Tatsache, dass diese Größe für die Gegenstelle bindend ist, was später noch eine Bedeutung hat.

Bevorzugterweise sollte man die MSS nach der lokalen MTU richten und diese auch so an die Gegenstelle übermitteln. Verwendet man aber stattdessen naiverweise die gespeicherte PMTU für diese Route, kann es passieren, dass diese wesentlich kleiner ist als die tatsächlich mögliche MSS. Diese Vorgangsweise ist in zwei Szenarien schädlich: zum einen kann es passieren, dass die Route zwischen den Gegenstellen asymmetrisch ist, und dass die PMTU in der einen Richtung größer ist als in der Entgegengesetzten. Im anderen Fall kann die Route zwar symmetrisch sein, aber die Begrenzung der MSS hindert die Gegenstelle zu einem späteren Zeitpunkt dran, den PMTUD Prozess erneut durchzuführen, um einen Anstieg der PMTU zu detektieren. Da die MSS für die Gegenstelle bindend ist, darf diese laut TCP Spezifikation keine größeren Pakete schicken, obwohl es ja durchaus sein könnte dass sowohl der Pfad, als auch der Empfänger in der Lage wären, mit größeren Segmenten zu arbeiten.

3.1.3 Stretch ACKS

Ein ACK wird in einer TCP Verbindung vom Empfänger gesendet, um dem Sender anzuzeigen, dass seine Pakete erfolgreich angekommen sind. Allerdings wird oft nicht für jedes einzelne Paket ein ACK geschickt, sondern für mehrere, sodass die Verbindung nicht unnötig belastet wird. Implementiert man das Versenden der ACKs in seinem TCP-Stack auf eine einfache Art und Weise, provoziert man das Problem der so genannten “Stretch ACKs”, das im Zusammenhang mit PMTUD auftritt. Angenommen man legt fest, dass alle zwei Segmente (MSS) ein ACK versendet wird. Wenn dann die Gegenstelle mittels PMTUD eine relativ kleine PMTU festgestellt hat, und danach die Größe seiner Pakete richtet, dann ist das Verhältnis zwischen PMTU und MSS relativ groß, und es dauert eine ganze Weile, bis genügend Daten zusammengekommen sind und ein Segment sozusagen aufgefüllt ist, woraufhin erst das ACK gesendet wird. Durch diese Unregelmäßigkeit der eintreffenden ACKs muss der Empfänger immer wieder auf die Bestätigung warten, bis er die nächsten Pakete verschicken kann. Das wiederum erzeugt eine unregelmäßige Transfer-Auslastung, was eine vorhandene Leitung nicht optimal ausnützen kann.

3.2 Vorteile von PMTU-Options

Ziel dieses Verfahrens ist es sowohl konzeptionelle Schwächen von PMTUD zu beseitigen, als auch über Probleme hinwegzuhelfen, die beim tatsächlichen Einsatz auftreten. Deswegen wird hier kurz auf mögliche Vorteile und deren Anwendbarkeit eingegangen.

3.2.1 Kein Paketverlust

Da die mit den IP-Optionen ermittelte PMTU eine Obergrenze für das weiterhin zu verwendende PMTUD Verfahren liefert, ist es bei Unterstützung durch den Router mit der kleinsten MTU möglich, den Link optimal auszunutzen, ohne durch PMTUD verursachten Paketverlust hinzunehmen.

3.2.2 Kein Black Hole Detection

Auch das so genannte “Black Hole Detection” Problem, bei dem der PMTUD Prozess aufgrund verworfener ICMP Pakete misslingt, und dadurch die Verbindung scheitert kann umgangen werden, da das Verfahren im Optimalfall nicht auf ICMP Pakete angewiesen ist. Falls der Router mit der kleinsten MTU die IP-Option setzt, die beim Sender als Obergrenze für die Paketgröße fungiert, werden trotz größtmöglicher PMTU niemals Pakete generiert die verworfen werden müssen. Und ebenso werden natürlich auch keine ICMP Nachrichten generiert und verschickt, die verloren gehen könnten.

3.2.3 Geringere Belastung am Bottleneck

Die Arbeit des Routers, der an einem Link mit kleiner MTU hängt, wird erleichtert. Normalerweise müsste er eine Menge an “ICMP Fragmentation needed” Nachrichten generieren, was für einen Router eine teure Operation sein kann: es muss der Speicher für das Paket alloziert werden, die Felder müssen initialisiert werden, etc. Abgesehen davon wird durch die ICMP Pakete zusätzlicher Verkehr im Netzwerk erzeugt, der ebenfalls an der Bandbreite zehrt. Im Falle dass der Rückweg überlastet ist und Stau auftritt kann es auch noch passieren dass das Paket verloren geht. Diese Probleme werden durch die PMTU-Optionen vermieden.

3.3 Nachteile von PMTU-Options

3.3.1 Slow Path

IP-Optionen erfordern eine spezielle Behandlung durch den Router, weswegen sie oft in den so genannten “Slow Path” wandern. Das ist ein Teil des Routers, bei dem die Pakete nicht durch spezielle Hardware weitergeleitet werden, sondern bei dem mit Hilfe von Software die Optionen ausgewertet werden, was natürlich die Verarbeitung verlangsamt. Hier sei allerdings auf die in [1] dokumentierten Messungen verwiesen, die besagen, dass im Durchschnitt das Delay eines Paketes mit Optionen nur etwa 7% größer ist. Ein anderer möglicher Effekt der in diesem Kontext allerdings durchaus auch auftreten könnte, wäre dass Pakete nicht in geordneter Reihenfolge ankommen, da das Paket, das die Option enthält, im Gegensatz zu den übrigen verspätet ankommt.

3.3.2 Nur sinnvoll wenn Sender, Router und Empfänger mitspielen

Da IP-Optionen im Internet zum Teil nur unzureichend unterstützt werden, und viele Router, Firewalls oder Hosts Pakete die mit solchen Optionen versehen sind verwerfen, begrenzt das den Einsatz des beschriebenen Verfahrens auf spezielle Fälle, bzw. unterstreicht den experimentellen Status der Spezifikation.

3.3.3 Pakete gehen verloren

Pakete die IP-Optionen enthalten gehen manchmal aus unerklärlichen Gründen verloren. Vor allem im öffentlichen Internet kann man sich also nicht zu hundert Prozent

darauf verlassen, dass so ein Paket ordnungsgemäß zugestellt wird. Das liegt vermutlich daran, dass auch ältere Hard-, bzw. Software im Einsatz ist, oder dass restriktiv konfigurierte Firewalls entsprechende Pakete unterdrücken. Das legt es nahe, PMTU-Optionen nur für spezielle Pfade einzusetzen, was ja auch der vorige Punkt deutlich macht.

3.3.4 Router Belastung

Wie bereits erwähnt entlastet das Verfahren zwar den Router am Bottleneck, im Gegensatz zu PMTUD steigt aber auch die Belastung auf den Routern entlang des Pfades. Das ergibt sich aus der schon genannten Behandlung des Pakets im Slow-Path, die ja alle Router entlang des Pfades trifft. Das wird allerdings durch die wegfallenden ICMP Pakete relativiert, sowie die nicht mehr anfallenden Re-Transmits der verworfenen Pakete, sodass insgesamt weniger Verkehr auftritt.

3.4 Tunnels

Ein weitere günstige Einsatzmöglichkeit ergibt sich bei der Verwendung in Kooperation mit einem Netzwerk Tunnel, bei dem beispielsweise ein Paket in ein anderes IP-Paket eingebettet wird. Hier kann es schnell passieren, dass das ursprüngliche Paket und der zusätzliche Header des Tunnels die MTU des Links überschreiten, was dann entweder durch die Tunnel-Endpunkte mittels PMTUD oder durch Fragmentierung der getunnelten Pakte trotz gesetztem “Don’t Fragment” Bit gelöst werden kann. Setzt man hier an, und kopiert die PMTU-Optionen in den äußeren Header, ergibt sich eine einfache Möglichkeit, den Tunnel mittels passender Pakete optimal zu nutzen.

4 Spezifikation

Eine mögliche Lösung ist in einem Internet-Draft spezifiziert [8]. Dort ist beschreiben, wie das zuvor erwähnte Verfahren funktionieren kann.

4.1 PMTUD mit IP-Optionen

Das grundsätzliche Prinzip wurde bereits erläutert, deswegen wird hier nur noch auf Details aus [8] eingegangen. Aufgebaut ist die PMTU IP Option wie in Abbildung 4 dargestellt.

Das ist die Option, die auf jeden Fall benötigt wird, da sie für den eigentlichen Test eingesetzt wird, während die Antwort ja auch in einer höheren Netzwerkschicht transportiert werden kann. Der Typ der Option ist Nummer 11, das entspricht dem in [6] festgelegten Typ für eine Probe-MTU Option, allerdings ist die gesamte Länge auf 8 Byte angewachsen, was durch die beiden zusätzlichen Felder TTL-Check und MTU-Nounce ausgefüllt wird. Die zusätzlichen Konsistenzchecks aufbauend auf diesen Feldern wurden in der hier behandelten Proof-Of-Concept Implementierung nicht realisiert, es wird aber trotzdem kurz die Funktionsweise erklärt.

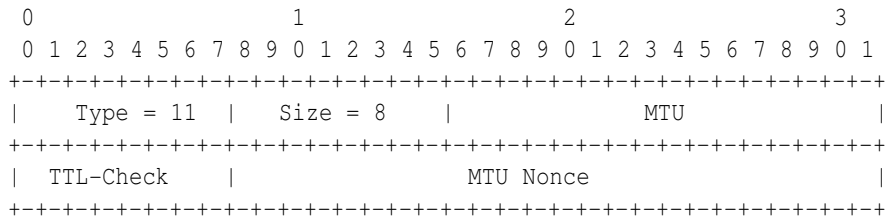

Abbildung 4: PMTUD-Option

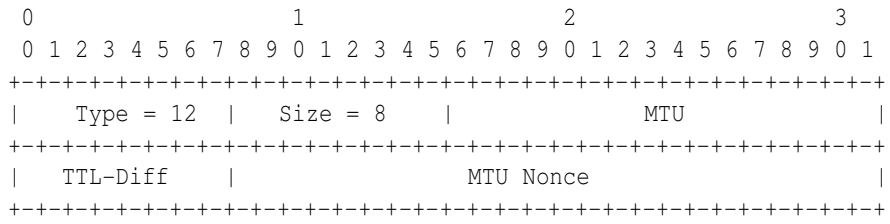

Abbildung 5: PMTUD-Antwort

Zum MTU Feld: hier schreibt der Sender die MTU seines ausgehenden Links hinein. Das TTL-Check Feld wird mit einer zufälligen Zahl initialisiert, ebenso das MTU-Nonce Feld. TTL-Check dient dazu, festzustellen ob alle Router entlang des Pfades die Optionen verarbeiten konnten. Dazu speichert der Sender intern sowohl den Wert des TTL-Check Feldes, als auch die Differenz mit der TTL. Jeder Router der dann die Option verarbeitet zieht 1 von dem TTL-Check wert ab, bevor er das Paket weiterleitet. Sobald der Sender dann die Antwort mit den entsprechenden Werten zurückerhalten hat, kann er daraus seine Schlüsse ziehen.

Das MTU-Nonce Feld dient dazu, böswillige Veränderungen am Paket zu erkennen, was insbesondere darauf abzielt, zu verhindern dass ein Angreifer die Tatsache verschleiern will, dass die Option von einem Router aktualisiert wurde. Deswegen ist ein Router der die Option verändert angehalten, dieses Feld auf Null zu setzen. Für einen späteren Vergleich muss der Sender natürlich auch diesen Wert speichern.

Des weiteren wird natürlich ein Antwort-Paket benötigt, das ebenfalls als IP-Option geschickt werden kann, was allerdings angesichts der Tatsache vermieden werden sollte, dass dadurch unnötigerweise auch die Router am Rückweg belastet werden. Es wurde dennoch dieser Weg für die hier erläuterte Implementierung gewählt, da bei einer Interaktion zwischen den verschiedenen Netzwerk-Layern im Linux-Kernel mit wesentlich komplexeren Fallstricken zu rechnen ist. Demzufolge beschränken sich auch die Erläuterungen auf diesen Typ der Antwort, die übrigen werden in [8] ausführlich erklärt. Die Antwort ist wie zu erwarten praktisch identisch zu der vorherigen aufgebaut, wie man in Abbildung 5 sehen kann.

Der Unterschied besteht vorrangig darin, dass der Typ auf 12 gesetzt ist. Initialisiert werden sämtliche Felder mit den Werten die der Empfänger mit der Test-Option erhalten hat, bis auf die TTL-Diff Option, die mit der Differenz des erhaltenen TTL-Check Wertes und des TTL (Time to Live) Wertes im IP-Header gefüllt wird. Ein Router der

einem solchen Paket begegnet hat natürlich nichts zu tun, das Paket sollte völlig unverändert weitergeleitet werden.

4.2 Implementierung

Um ein optimales Verhalten zu gewährleisten ist es nötig den erhaltenen PMTU Wert korrekt zu behandeln. Es wurde bereits erwähnt, dass er im wesentlichen als Obergrenze für das klassische PMTUD Verfahren einzusetzen ist. Sollte aber bereits ein PMTU Wert in den Routing-Tabellen stehen, der aufgrund eines früheren PMTUD Vorgangs ermittelt wurde, sollte man vom Erhöhen der PMTU absehen, da möglicherweise ein Router die Option nicht verarbeitet hat. Es lässt sich allerdings feststellen ob alle Router entlang des Pfades die Option unterstützt haben, indem man den TTL-Diff Wert aus der Antwort mit dem gespeicherten vergleicht. Stimmt er überein, kann man davon ausgehen, das der ermittelte PMTU Wert nicht nur eine obere Grenze ist, sondern der tatsächlichen PMTU entspricht. Sollte allerdings gar keine Antwort innerhalb eines festgelegten Zeitrahmens auftreten, ist davon auszugehen dass der Empfänger keine Unterstützung für die Option bietet, und man kann stattdessen den klassischen PMTUD Prozess durchführen.

Allerdings kann ja ein Paket auch einfach verloren gegangen sein, weswegen es Sinn macht die Test-Option mehrmals zu senden, um die Belastung der Router gering zu halten allerdings nur in ausreichend großzügig gewählten Abständen. Dies sollte auch dann noch geschehen, wenn die PMTU bereits feststeht, um bei Verbindungen die über einen längeren Zeitraum bestehen zu gewährleisten, dass Änderungen im Pfad, die eine Erhöhung der PMTU zur Folge haben, erkannt werden.

5 Im Linux Netzwerk Code

Der Netzwerkstack im Linux-Kernel ist auf Basis so genannter Sockets und Socket-Buffers implementiert. Ein Socket steht dabei für eine Verbindung, und ein Socket-Buffer für ein Paket, das über eine Verbindung verschickt, empfangen oder weitergeleitet wird. Diese beiden Datenstrukturen sind in den Dateien `include/linux/skbuff.h` und `include/net/sock.h` definiert. Im Laufe der Implementierung wurden dort einige Attribute hinzugefügt, um den momentanen Status des PMTUD Vorgangs zu speichern. Um das erläuterte Verfahren zu implementieren, sind einige Änderungen im Netzwerk Code des Linux-Kernels nötig. Deswegen wird kurz darauf eingegangen, welche Bereiche betroffen sind, und wie diese funktionieren. Zuerst werden die beteiligten Datenstrukturen behandelt, und anschließend die Funktionen.

5.1 Datenstrukturen

5.1.1 Socket-Buffer

`include/linux/skbuff.h`

Ein skbuff entspricht exakt einem Datenpaket das später ins Ethernet versendet wird. Ein Socket Buffer enthält alle Header der unterschiedlichen Netzwerkschichten,

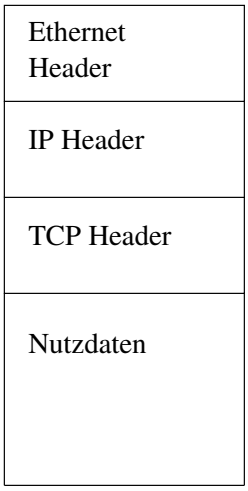

Abbildung 6: Socket-Buffer

sowie die Nutzdaten; siehe [3]. Um eine möglichst effiziente Verarbeitung der Pakete im Kernel zu ermöglichen, ist der Socket-Buffer so aufgebaut, dass die Header einer tiefer liegenden Protokollschicht direkt vorne angehängt werden können (siehe Abbildung 6).

Abgesehen von dem genauen Abbild des Pakets enthält ein Socket-Buffer noch einiges an Zusatzinformationen, beispielsweise zu welchem Socket er gehört, welche IP-Optionen gesetzt sind, etc.

5.1.2 Socket

`include/net/sock.h`

Informationen über eine Verbindung, welche Art von Verbindung der Socket darstellt, z.B. TCP, über welchen Port das ganze läuft, wer die Gegenstelle ist, etc. werden in einem so genannten Socket gespeichert, siehe [2]. Diese Datenstruktur ist wichtig, weil hier Informationen über die PMTUD gespeichert werden, wie z.B. ob der Vorgang bereits abgeschlossen wurde und was die ermittelte PMTU ist.

5.2 Funktionen

5.2.1 ip_forward_finish

Zu finden in der Datei `net/ipv4/ip_forward.c`.

Diese Funktion wird aufgerufen, wenn feststeht, dass ein IP-Paket weitergeleitet werden muss; hier ist also eine möglicher Stelle, um die Routerfunktion, d.h. das Ändern der MTU-Option, zu implementieren. Dabei muss natürlich auch die Prüfsumme des Pakets aktualisiert werden, was mit Hilfe der Funktion `ip_send_check` gemacht wird.

5.2.2 ip_rcv_finish

Zu finden in der Datei `net/ipv4/ip_input.c`.

Hier werden die IP-Optionen eingehender Pakete verarbeitet. Das heißt, es wird überprüft ob das Paket Optionen enthält, und falls ja, werden die entsprechenden Flags des Socket-Buffers gesetzt. Dies ermöglicht es, später direkt auf die Eigenschaften des Pakets zugreifen zu können, ohne erneut alle Optionen parsen zu müssen. Das alles ist für zwei Dinge nützlich: zum einen, wenn ein PMTU-Paket von der Gegenstelle eintrifft, und zum anderen, wenn man die Antwort der Anfrage erhält, die man selbst gestellt hat.

5.2.3 ip_queue_xmit

Zu finden in der Datei `net/ipv4/ip_output.c`.

Wenn ein Paket den Rechner verlässt, wird diese Funktion aufgerufen, damit ist es möglich, die gewünschten IP-Optionen hier einzufügen. Wie zuvor gibt es wieder zwei Szenarien, zum einen das Versenden der eigenen Anfrage, und zum anderen die Antwort auf eine zuvor erfolgte Abfrage der Gegenseite.

5.2.4 tcp_v4_rcv

Zu finden in der Datei `net/ipv4/tcp_ipv4.c`.

In dieser Funktion passiert die Zuordnung eines eingehenden Pakets zum passenden Socket. Deswegen macht es an dieser Stelle Sinn, allfällige IP-Optionen die ein ankommendes Paket hat, in die Socket Datenstruktur zu übertragen, um z.B. später die entsprechende Antwort zu schicken.

5.2.5 tcp_sync_mss

Zu finden in der Datei `net/ipv4/tcp_output.c`.

Um die aktuelle TCP-MSS zu ermitteln wird diese Funktion aufgerufen, um die Größe des nächsten TCP-Pakets zu ermitteln. Damit ist hier eine wichtige Stelle, um z.B. Platz zu schaffen damit die PMTU-Option zum Einsatz kommen kann, aber auch um die durch das Verfahren ermittelte PathMTU zu berücksichtigen und keine größeren Pakete zuzulassen.

6 Implementierung

Dieser Patch ist nur ein Proof-Of-Concept, kein einsetzbarer Produktions-code. die Behandlung der IP-Optionen ist nicht korrekt sondern funktioniert nur solange keine anderen IP-Optionen im Einsatz sind. Das Herabsetzen der TCP-Paket Größe (MSS) wird immer durchgeführt, anstatt nur wenn nötig, und zusätzliche Features aus [8] wurden überhaupt nicht implementiert. Der Code soll lediglich das Prinzip zeigen, und einfache Experimente und Benchmarks erlauben.

Das Verfahren kennt mehrere Rollen, die alle zusammenspielen müssen. Dort wo die IP-Pakete gesendet werden, muss die Option in den IP-Header eingebaut werden,

beim Empfang eines Pakets mit der Option muss eine Antwort geschickt werden, und für ein weiterzuleitendes Paket wird nötigenfalls der MTU-Wert decrementiert. Im folgenden werden die Änderungen beschrieben, die vorgenommen wurden, um das Verfahren zu implementieren.

In der Socket-Buffer (skbuff) Datenstruktur wurden vier Werte hinzugefügt:

```
unsigned int send_pmtu_opt_reply;
unsigned int pmtu_opt_reply_mtu;
unsigned int max_pmtu_option;
unsigned int got_pmtu_option_reply;
```

Der Socket selbst (sock) hat folgende Flags erhalten.

```
unsigned int max_pmtu_option;
unsigned int got_pmtu_option_reply;
unsigned int send_no_more_pmtu_options;
unsigned int send_pmtu_opt_reply;
unsigned int pmtu_opt_reply_mtu;
```

Diese beiden werden zuerst im skbuff ermittelt, und anschließend an den Socket weitergegeben. Dann wurden als erstes die Funktion zum Senden der IP-Option implementiert, und zwar in der Funktion ip_queue_xmit. Dazu wurde einfach der IP-Header etwas verlängert, und die entsprechende Option eingefügt.

```
if (!sk->send_no_more_pmtu_options) {
    iph->ihl+=8>>2;
    unsigned short pmtu_options[8] = {0};
    pmtu_options[0] = htons(0x0B08);
    pmtu_options[1] = htons(rt->u.dst.dev->mtu);
    pmtu_options[2] = 0;
    pmtu_options[3] = 0;
    memcpy( skb->nh.raw+sizeof(struct iphdr), pmtu_options, 8);
    sk->send_no_more_pmtu_options = 1;
}
```

Wie man sieht ist auch die MTU des übertragenden Links enthalten. Dadurch wird mit dem ersten Paket, das den Stack verlässt, die Option mitgegeben, und die Gegenstelle kann entsprechend darauf reagieren.

Der folgende Code implementiert die Forwarding Funktion, er wurde in der erwähnten Stelle eingefügt:

```
if (opt->optlen > 0) {
    unsigned short out_mtu = ((struct rtable*)skb->dst)->u.dst.dev->mtu;
    unsigned char *opt_ptr = skb->nh.raw + sizeof(struct iphdr);
    if ( *opt_ptr == 0x0B ) {
        unsigned short opt_mtu = ntohs(*((unsigned short*)(opt_ptr+2)));
        if ( out_mtu < opt_mtu) {
            *((unsigned short*)(opt_ptr+2)) = htons(out_mtu);
            ip_send_check(skb->nh.iph);
        }
    }
}
```

Hier wird überprüft ob die Länge der Optionen größer als 0 ist, und falls ja, wird überprüft ob es sich um die gesuchte Option handelt. Je nachdem ob dies der Fall ist, und ob die MTU des nächsten Links kleiner ist, wird der Wert im Paket entsprechend modifiziert, sowie die Header-Checksumme aktualisiert.

Was nun noch fehlt, ist dass der Kernel auch die passende Antwort auf eine Anfrage schickt; dazu muss als erstes überprüft werden ob und welche Optionen ein empfangenes Paket enthält. Das passiert wie bereits erwähnt in ip_rcv_finish Funktion, wo folgender Code eingefügt wurde:

```
unsigned char* opt_ptr;
opt_ptr = skb->nh.raw + sizeof(struct iphdr);
if (*opt_ptr == 0x0B ) {
    unsigned short reply_mtu = htons(*((unsigned short*)(opt_ptr+2)));
    skb->send_pmtu_opt_reply = 1;
    skb->pmtu_opt_reply_mtu = reply_mtu;
}
```

Hier werden einfach nur zwei Werte im Socket Buffer gesetzt, zum einen, dass eine Antwort gesendet werden soll, und zum anderen der Wert der Antwort. Diese Werte werden dann in der Funktion tcp_v4_rcv in die Datenstruktur des Sockets kopiert.

```
if (skb->send_pmtu_opt_reply) {
    sk->send_pmtu_opt_reply = skb->send_pmtu_opt_reply;
    sk->pmtu_opt_reply_mtu = skb->pmtu_opt_reply_mtu;
}
```

Sobald das nächste Paket den Rechner verlässt, wird in `ip_queue_xmit` die Antwort Option wieder angehängt, und das Flag zurückgesetzt.

```
if (skb->sk->send_pmtu_opt_reply) {
    iph->ihl+=8> >2;
    unsigned short pmtu_reply_options[4] = {0};
    pmtu_reply_options[0]=htons(0x0C08);
    pmtu_reply_options[1]=htons(skb->sk->pmtu_opt_reply_mtu);
    memcpy(skb->nh.raw+sizeof(struct iphdr), pmtu_reply_options, 8);
    skb->sk->send_pmtu_opt_reply=0;
}
```

Was dann passiert, ist die Ankunft des Antwort-Pakets am anderen Rechner, die analog zu den bereits erklärten Vorgängen funktioniert, und nahezu identisch implementiert wurde. Allerdings muss nun auch die erhaltene Antwort beim Ermitteln der nächsten Paketgröße in Betracht gezogen werden; das erledigt folgendes Konstrukt, das in die Funktion `tcp_sync_mss` eingebunden wurde.

```
if (tp->got_pmtu_option_reply) {
    pmtu = tp->max_pmtu_option;
} else {
    pmtu = 576;
}
```

Dadurch wird entweder die PMTU Antwort verwendet, oder die erlaubte minimale Größe.

7 Dumps

Unten stehende Tabelle zeigt ein Beispiel für eine Verbindung mit PMTUD mit Optionen.

Nr.	Size	Sender	Reciever	pmtu-Option
3	82	jupiter	zeus	pmtu-ask 1500
8	74	zeus	jupiter	pmtu-reply 1111
	...			
40	1117 (IP-Size: 1103)	jupiter	zeus	
41	1117 (IP-Size: 1103)	jupiter	zeus	

Dieses Beispiel wurde auf einer Test-Strecke ermittelt, die aus drei Rechnern besteht, von denen einer als Router fungiert. Rechner “jupiter” ist über eine Verbindung mit MTU 1500 am Router “god” angebunden, und Rechner “zeus” über eine Verbindung mit MTU 1111 zu Router “god”. Die Packetgröße von 1103 ergibt sich daraus, dass 8 Byte für die IP-Option abgezogen werden.

Darunter ein Beispiel für eine Verbindung über die selbe Konfiguration, allerdings ohne PMTUD Optionen. Hier sieht man, dass zwei Pakete mit zu großer MTU geschickt werden; diese werden verworfen, es wird die gewünschte Fehlermeldung generiert, und erst danach werden die Pakete mit der optimalen Größe geschickt.

Nr	Size	Sender	Reciever	misc.
	. . .			
6	1514	jupiter	zeus	lost
7	590	god	jupiter	ICMP Dest. unreachable
8	1514	jupiter	zeus	lost
9	590	god	jupiter	ICMP Dest. unreachable
10	1125 (IP-Size: 1111)	jupiter	zeus	
	. . .			

8 Kernel-Patch

Der folgende Patch wurde auf Basis der Linux-Version 2.4.26 erstellt.

```
diff -u -r linux-2.4.26/include/linux/skbuff.h linux-2.4.26-pmtu-all-pat/include/linux/skbuff.h
--- linux-2.4.26/include/linux/skbuff.h 2004-04-14 15:05:40.000000000 +0200
+++ linux-2.4.26-pmtu-all-pat/include/linux/skbuff.h 2004-06-09 16:28:26.000000000 +0200
@@ -196,6 +196,10 @@
unsigned char *data; /* Data head pointer */
unsigned char *tail; /* Tail pointer */
unsigned char *end; /* End pointer */
+ unsigned int send_pmtu_opt_reply;
+ unsigned int pmtu_opt_reply_mtu;
+ unsigned int max_pmtu_option;
+ unsigned int got_pmtu_option_reply;
void (*destructor)(struct sk_buff *); /* Destruct function */
#ifdef CONFIG_NETFILTER
diff -u -r linux-2.4.26/include/net/sock.h linux-2.4.26-pmtu-all-pat/include/net/sock.h
--- linux-2.4.26/include/net/sock.h 2004-04-14 15:05:40.000000000 +0200
+++ linux-2.4.26-pmtu-all-pat/include/net/sock.h 2004-06-09 16:29:01.000000000 +0200
```

```
@@ -432,6 +432,8 @@
__u32 frto_highmark; /* snd_nxt when RTO occurred */
unsigned long last_synq_overflow;
+ unsigned int max_pmtu_option;
+ unsigned int got_pmtu_option_reply;
/* TCP Westwood structure */
struct {
@@ -631,6 +633,9 @@
int rcvlowat;
long rcvtimeo;
long sndtimeo;
+ unsigned int send_no_more_pmtu_options;
+ unsigned int send_pmtu_opt_reply;
+ unsigned int pmtu_opt_reply_mtu;
#ifdef CONFIG_FILTER
/* Socket Filtering Instructions */
diff -u -r linux-2.4.26/net/ipv4/ip_forward.c linux-2.4.26-pmtu-all-pat/net/ipv4/ip_forward.c
--- linux-2.4.26/net/ipv4/ip_forward.c 2001-04-12 21:11:39.000000000 +0200
+++ linux-2.4.26-pmtu-all-pat/net/ipv4/ip_forward.c 2004-06-09 16:28:26.000000000 +0200
@@ -46,6 +46,19 @@
struct ip_options * opt = &(IPCB(skb)->opt);
IP_INC_STATS_BH(IpForwDatagrams);
+ if (opt->optlen > 0) {
+ unsigned short out_mtu = ((struct rtable*)skb->dst)->u.dst.dev->mtu;
+ unsigned char *opt_ptr = skb->nh.raw + sizeof(struct iphdr);
+ printk(KERN_INFO "Optionen Paket\n");
+ if ( *opt_ptr == 0x0B ) {
+ printk(KERN_INFO "PMTU Paket\n");
+ unsigned short opt_mtu = ntohs(*((unsigned short*)(opt_ptr+2)));
+ if ( out_mtu < opt_mtu) {
+ *((unsigned short*)(opt_ptr+2))=htons(out_mtu);
+ ip_send_check(skb->nh.iph);
+ }
+ }
+ }
if (opt->optlen == 0) {
#ifdef CONFIG_NET_FASTROUTE
diff -u -r linux-2.4.26/net/ipv4/ip_input.c linux-2.4.26-pmtu-all-pat/net/ipv4/ip_input.c
--- linux-2.4.26/net/ipv4/ip_input.c 2002-08-03 02:39:46.000000000 +0200
+++ linux-2.4.26-pmtu-all-pat/net/ipv4/ip_input.c 2004-06-09 16:28:26.000000000 +0200
@@ -327,6 +327,8 @@
st[(idx>>16)&0xFF].i_bytes+=skb->len;
}
#endif
+ skb->send_pmtu_opt_reply = 0;
+ skb->got_pmtu_option_reply = 0;
if (iph->ihl > 5) {
struct ip_options *opt;
@@ -342,7 +344,21 @@
if (skb_cow(skb, skb_headroom(skb)))
goto drop;
iph = skb->nh.iph;
-
+ unsigned char* opt_ptr;
+ opt_ptr = skb->nh.raw + sizeof(struct iphdr);
+ if (*opt_ptr == 0x0B ) {
+ unsigned short reply_mtu = htons(*((unsigned short*)(opt_ptr+2)));
+ printk(KERN_INFO "Option empfangen: MTU_Size: %d\n", reply_mtu);
+ skb->send_pmtu_opt_reply = 1;
+ skb->pmtu_opt_reply_mtu = reply_mtu;
+ }
+ if ( *opt_ptr== 0x0C ) {
+ unsigned short reply_mtu = htons(*((unsigned short*)(opt_ptr+2)));
+ printk(KERN_INFO "Option Antwort empfangen: %d\n", reply_mtu);
+ skb->got_pmtu_option_reply = 1;
+ skb->max_pmtu_option = reply_mtu;
+ }
+
```

```
if (ip_options_compile(NULL, skb))
goto inhdr_error;
diff -u -r linux-2.4.26/net/ipv4/ip_output.c linux-2.4.26-pmtu-all-pat/net/ipv4/ip_output.c
--- linux-2.4.26/net/ipv4/ip_output.c 2003-11-28 19:26:21.000000000 +0100
+++ linux-2.4.26-pmtu-all-pat/net/ipv4/ip_output.c 2004-06-09 16:28:26.000000000 +0200
@@ -381,7 +381,11 @@
goto no_route;
/* OK, we know where to send it, allocate and build IP header. */
- iph = (struct iphdr *) skb_push(skb, sizeof(struct iphdr) + (opt ? opt->optlen : 0));
+ if (skb->sk->send_pmtu_opt_reply || (!sk->send_no_more_pmtu_options)) {
+ iph = (struct iphdr *) skb_push(skb, sizeof(struct iphdr) + (opt ? opt->optlen : 0)+8);
+ } else {
+ iph = (struct iphdr *) skb_push(skb, sizeof(struct iphdr) + (opt ? opt->optlen : 0));
+ }
*((__u16 *)iph) = htons((4 << 12) | (5 << 8) | (sk->protinfo.af_inet.tos & 0xff));
iph->tot_len = htons(skb->len);
if (ip_dont_fragment(sk, &rt->u.dst) && !ipfragok)
@@ -399,6 +403,24 @@
iph->ihl += opt->optlen >> 2;
ip_options_build(skb, opt, sk->daddr, rt, 0);
}
+ if (skb->sk->send_pmtu_opt_reply) {
+ printk(KERN_INFO "Werde die Opt Antwort schicken: %d\n", skb->sk->pmtu_opt_reply_mtu);
+ iph->ihl+=8>>2;
+ unsigned short pmtu_reply_options[4] = {0};
+ pmtu_reply_options[0]=htons(0x0C08);
+ pmtu_reply_options[1]=htons(skb->sk->pmtu_opt_reply_mtu);
+ memcpy(skb->nh.raw+sizeof(struct iphdr), pmtu_reply_options, 8);
+ skb->sk->send_pmtu_opt_reply=0;
+ } else if (!sk->send_no_more_pmtu_options) {
+ iph->ihl+=8>>2;
+ unsigned short pmtu_options[8] = {0};
+ pmtu_options[0]=htons(0x0B08);
+ pmtu_options[1]=htons(rt->u.dst.dev->mtu);
+ pmtu_options[2]=0;
+ pmtu_options[3]=0;
+ memcpy( skb->nh.raw+sizeof(struct iphdr), pmtu_options, 8);
+ sk->send_no_more_pmtu_options = 1;
+ }
return NF_HOOK(PF_INET, NF_IP_LOCAL_OUT, skb, NULL, rt->u.dst.dev,
ip_queue_xmit2);
diff -u -r linux-2.4.26/net/ipv4/tcp_ipv4.c linux-2.4.26-pmtu-all-pat/net/ipv4/tcp_ipv4.c
--- linux-2.4.26/net/ipv4/tcp_ipv4.c 2004-04-14 15:05:41.000000000 +0200
+++ linux-2.4.26-pmtu-all-pat/net/ipv4/tcp_ipv4.c 2004-06-09 16:28:26.000000000 +0200
@@ -1759,6 +1759,17 @@
if (!sk)
goto no_tcp_socket;
+ if (skb->send_pmtu_opt_reply) {
+ sk->send_pmtu_opt_reply = skb->send_pmtu_opt_reply;
+ sk->pmtu_opt_reply_mtu = skb->pmtu_opt_reply_mtu;
+ printk(KERN_INFO "Antwort in Socket gespeichert\n");
+ }
+ if (skb->got_pmtu_option_reply) {
+ struct tcp_opt *tp = &sk->tp_pinfo.af_tcp;
+ tp->max_pmtu_option = skb->max_pmtu_option;
+ tp->got_pmtu_option_reply = skb->got_pmtu_option_reply;
+ }
+
process:
if(!ipsec_sk_policy(sk,skb))
goto discard_and_relse;
diff -u -r linux-2.4.26/net/ipv4/tcp_output.c linux-2.4.26-pmtu-all-pat/net/ipv4/tcp_output.c
--- linux-2.4.26/net/ipv4/tcp_output.c 2003-11-28 19:26:21.000000000 +0100
+++ linux-2.4.26-pmtu-all-pat/net/ipv4/tcp_output.c 2004-06-09 16:28:26.000000000 +0200
@@ -508,8 +508,16 @@
/* Calculate base mss without TCP options:
It is MMS_S - sizeof(tcphdr) of rfc1122
*/
```

```
+ if (tp->got_pmtu_option_reply) {
+ pmtu = tp->max_pmtu_option;
+ } else {
+ pmtu = 576;
+ }
mss_now = pmtu - tp->af_specific->net_header_len - sizeof(struct tcphdr);
+ if (sk->send_pmtu_opt_reply) {
+ mss_now-=8;
+ }
/* Clamp it (mss_clamp does not include tcp options) */
if (mss_now > tp->mss_clamp)
@@ -517,6 +525,7 @@
/* Now subtract optional transport overhead */
mss_now -= tp->ext_header_len;
+ mss_now -= 8;
/* Then reserve room for full set of TCP options and 8 bytes of data */
if (mss_now < 48)
```

Literatur

[1] M. Welzl, M. Rossi, "*On The Impact of IP Option Processing*", Preprint-Reihe des Fachbereichs Mathematik-Informatik (technical report), No. 15, 2003.

[2] M. Rio, M. Goutelle, T. Kelly, R. Hughes-Jones, J.P. Martin-Flatin und Y.T. Li: *A Map of the Networking Code in Linux Kernel 2.4.20* (DRAFT under review Work in Progress 23 March 2004)

[3] David A Rusling: *The Linux Kernel (http://www.tldp.org/LDP/tlk/tlk.html)*

[4] Information Sciences Institute University of Southern California, "*INTERNET PROTOCOL*", RFC 791, September 1981.

[5] J. C. Mogul und S. E. Deering, "*Path MTU discovery*", RFC 1191, November 1990.

[6] J.C. Mogul, C.A. Kent, C. Partridge und K. McCloghrie, "*IP MTU discovery options.*", RFC 1063, Juli 1988.

[7] K. Lahey, "*TCP Problems with Path MTU Discovery*", RFC 2923, September 2000.

[8] Michael Welzl, "*PMTU-Options: Path MTU Discovery Using Options*", Internet Draft, Februar 2003.

[9] M. Mathis, J. Heffner und K. Lahey, "*PATH MTU Discovery*", Internet Draft, Februar 2005.