Netzwerke I – Praktische Übungen

Übung 7: Zuverlässige Übertragung – Performance, Flow Control, Congestion Control

Aufgabe 7.1: Implementierung einer zuverlässigen Datenübertragung über UDP

Hinweis: Diese Übung umfasst nur diese eine – dafür jedoch etwas umfangreichere – Implementierungsaufgabe. Planen Sie bitte ausreichend Zeit zur Bearbeitung dieser Aufgabe ein. Beide Gruppenpartner müssen an der Implementierung dieser Aufgabe mitgewirkt haben, sonst ist keine Abnahme möglich.

Im Folgenden soll schrittweise eine zuverlässige Datenübertragung von Dateien über UDP implementiert werden. Hierzu sollen zwei eigenständige Anwendungen erstellt werden:

1. FileSender: Wird mit Dateinamen und Namen des Zielrechners als Parameter aufgerufen und versendet die angegebene Datei zuverlässig an den angegebenen Zielrechner.
2. FileReceiver: Wird ohne Parameter aufgerufen und wartet auf eingehende Dateiübertragungen. Empfangene Dateien werden lokal gespeichert. Nach dem Empfang einer Datei bleibt das Programm weiter aktiv und wartet auf die nächste eingehende Datenverbindung.

Die zuverlässige Übertragung der Daten über UDP soll mit Hilfe des Alternating Bit Protokolls auf Anwendungsschicht realisiert werden.

Gehen Sie bei der Umsetzung schrittweise vor:

a) Spezifizieren Sie geeignete Zustandsautomaten für FileSender und FileReceiver. Benennen Sie alle Zustände sinnvoll und kennzeichnen Sie gegebenenfalls Zustandsübergänge mit den entsprechenden Ereignissen bzw. Methodenaufrufen, die zum Zustandsübergang führen.

b) Spezifizieren Sie ein geeignetes Paketformat, welches alle notwendigen Elemente enthält, um Übertragungsfehler, Paketverluste und Reordering zu erkennen.

c) Implementieren Sie FileSender und FileReceiver **indem Sie den im ersten Aufgabenteil entwickelten Zustandsautomaten in ein Programm** überführen. Hierbei können Sie zunächst von der Annahme ausgehen, dass keine Übertragungsfehler, Paketverluste oder Reordering auftreten. Testen Sie ihre beiden Programme, indem Sie FileReceiver lokal auf dem Rechner starten und dann mit FileSender eine Datei an die localhost-Adresse senden.

d) Gehen Sie nun davon aus, dass Übertragungsfehler auftreten können. Erweitern Sie Ihre Implementierung so, dass Übertragungsfehler, Paketverluste und Reordering erkannt und mit Hilfe des Alternating Bit Protokolls behoben werden. (Hinweis: Falls Sie eine bestimmte, gegebene Zeit auf den Empfang eines Paketes warten wollen, können Sie dafür beispielsweise den aus dem vorherigen Aufgabenblatt bekannten Weg über den Socket-Timeout nutzen. Informieren Sie sich über die entsprechenden Methoden der Java Socket Klassen in der Java Dokumentation.)

e) Zum Test Ihres Programmes implementieren Sie jetzt eine einfache Simulation eines unzuverlässigen Kanals. Leiten Sie dazu empfangene UDP Pakete im FileReceiver durch eine Filter-Methode. Diese Methode realisiert die Simulation eines unzuverlässigen Kanals indem sie

   a. zufällig mit einer konfigurierbaren Wahrscheinlichkeit einen Bitfehler im Paket verursacht

   b. zufällig mit einer konfigurierbaren Wahrscheinlichkeit ein Paket verwirft

c. zufällig mit einer konfigurierbaren Wahrscheinlichkeit ein Paket dupliziert

f) Testen Sie, ob Ihr Programm eine Datei der Größe 1MB korrekt über den simulierten, fehlerbehafteten Kanal überträgt. Parameter: ein Paket wird mit p=0,1 verworfen, mit p=0,05 dupliziert und mit p=0,05 tritt ein Bitfehler im Paket auf. Prüfen Sie die empfangene Datei auf korrekte Übertragung, z.B. indem Sie ein Bild übertragen oder eine komprimierte Datei senden!

g) Installieren Sie FileSender und FileReceiver auf unterschiedlichen Laborrechnern. Welche Zeit brauchen Sie, um die Datei zu übertragen?

h) Vergleichen Sie die Datenübertragung mit Ihrem Programm mit einer Dateiübertragung über http und TCP. Unter welcher Bedingung würden Sie erwarten, dass Ihr Programm einen höheren Goodput erreicht als TCP?

Aufgabe 7.2: TCP Flow Control

Die Rechner A und B sind über einen 100MBit/s Netzwerkverbindung direkt verbunden. Es treten keine Paketfehler oder Paketverluste auf und es gibt genau eine TCP Verbindung zwischen beiden Rechnern. Über diese wird eine sehr große Datei von Rechner A zu Rechner B übertragen.

Rechner A liest die Daten von der Festplatte ein und kann sie mit einer Geschwindigkeit von 45 Mbit/s in den TCP-Socket einspeisen. Rechner B, der die empfangenen Daten auf einen USB-Stick speichert, liest sie jedoch nur mit einer Rate von 20 Mbit/s aus dem Socket (und damit aus dem TCP Empfangspuffer) aus.

a) Beschreiben Sie in Stichworten die Wirkung von TCP Flow Control in diesem Beispiel.
b) Welche Senderate stellt sich langfristig bei Rechner A ein?
c) Was würde passieren, wenn der Rechner B für eine Zeit von mehreren Sekunden keine Daten aus dem Socket liest, z.B. weil ein Schreibproblem auf dem USB-Stick auftritt, danach jedoch den Empfangspuffer leert?

Aufgabe 7.3: TCP Congestion Control

Gehen Sie wiederum von der Übertragung einer sehr großen Datei zwischen zwei Rechnern A und B über eine fehlerfreie TCP-Verbindung aus. Es treten keine Paketverluste auf und die Round Trip Time (RTT) sei konstant 10 ms. Als Maximum Segment Size (MSS) seien 1000 Byte angenommen.

a) Gehen Sie davon aus, dass eine Standard-TCP Version „TCP Reno“ genutzt wird. Wie lange dauert es, bis das Congestion Window (Überlastfenster) eine Größe von mindestens 8 MSS erreicht hat? Wie viele Daten werden vom Start der Verbindung bis zum Zeitpunkt 50 ms danach übertragen?
b) Gehen Sie jetzt alternativ zu a) davon aus, dass eine TCP Version *ohne Slow Start* genutzt wird, die das Congestion Window um 1 MSS/cwnd pro empfangenem ACK vergrößert. Wie lange dauert es jetzt bis das Congestion Window die Größe 8 MSS erreicht? Wie viele Daten werden nun vom Start der Verbindung bis zum Zeitpunkt 50 ms danach übertragen?

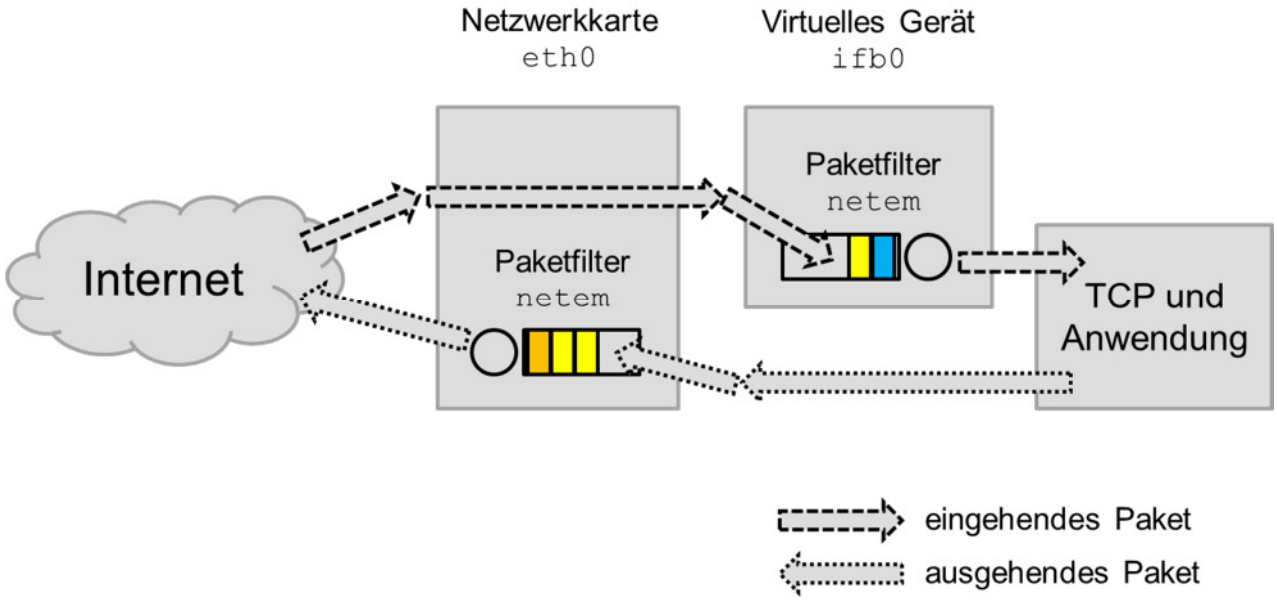
Aufgabe 7.4: TCP Verhalten unter unterschiedlichen Übertragungsbedingungen

In dieser Aufgabe soll das Verhalten von TCP unter unterschiedlichen Übertragungsbedingungen untersucht werden. Da wir – um die Übertragungsbedingungen zu ändern – nicht den Quellcode von TCP im Betriebssystem ändern wollen um einen Paketfilter einzubauen (so wie Sie es im Aufgabenblatt 7 innerhalb Ihres Java Programms getan haben), nutzen wir einen Paketfilter vom Betriebssystem selbst.

Dieser ist auf Ihrer Virtuellen Maschine bereits vorhanden und nennt sich „`netem`“ (Network Emulation). Über diesen können Sie verschiedenste Verzögerungen, Paketfehler, Paketumordnen usw. emulieren und die echte, im Betriebssystem vorhandene TCP Implementierung darüber testen.

Es gibt hierbei jedoch folgende Herausforderung: Einen Paketfilter können Sie generell nur für die von einem Netzwerkgerät versendeten Pakete einsetzen (Warum?). Um empfangene Pakete zu modifizieren nutzt man daher folgenden Trick: Die über die Netzwerkkarte eingehenden Pakete werden zunächst an ein „virtuelles Netzwerkgerät“ (Intermediate Functional Block, IFB) weitergeleitet. Auf diesem kann dann für ausgehende Pakete der Paketfilter installiert werden. Erst danach werden sie vom virtuellen Netzwerkgerät ans Betriebssystem und den Empfangsprozess ausgeliefert.

Der Weg eines Paketes sieht damit folgendermaßen aus:



Um das oben abgebildete Szenario auf Ihrer Virtuellen Maschine umzusetzen, gehen Sie bitte in folgenden Schritten vor:

1. **Einrichtung des virtuellen Gerätes ifb0**
   Wechseln sie zunächst – um nicht jedes Mal erneut „sudo“ eingeben zu müssen – zum Nutzer root. Dazu geben Sie in einem Kommandozeilen-Fenster den Befehl „sudo -i“ ein. Geben Sie danach folgende Befehle zur Konfiguration des virtuellen Gerätes ein:

   a. `modprobe ifb`
      Hiermit laden Sie einen Treiber (unter Linux „Kernel Modul“ genannt) für das virtuelle Gerät.

   b. `ip link set dev ifb0 up`
      Das Gerät ifb0 wird aktiviert.

2. Aufsetzen der Filter für die Netzwerk-Emulation

a. *Achtung: Bitte prüfen Sie zunächst durch Eingabe von „ifconfig“ welchen Namen die auf Ihrer virtuellen Maschine aktive Netzwerkkarte hat (in der Regel eth0, eth1 oder eth2).*
`tc qdisc add dev eth0 ingress`
Das Kommando “tc” (steht für Traffic Control) installiert hier eine Warteschlangenverwaltung (Queuing Discipline, abgekürzt qdisc) für eigehende Pakete auf der ersten Netzwerkkarte. Diese trägt hier im Beispiel den Namen eth0.

b. `tc filter add dev eth0 parent ffff: protocol ip u32 match u32 0 0 flowid 1:1 action mirred egress redirect dev ifb0`
Diesen Befehl bitte komplett hintereinander eingeben. Er bewirkt die Weiterleitung der eingehenden Pakete von der Netzwerkkarte eth0 auf das virtuelle Gerät `ifb0` und entspricht damit dem Pfeil zwischen `eth0` und `ifb0` in der Grafik oben.

c. `tc qdisc add dev ifb0 root handle 1: tbf rate 1024kbit buffer 1600 limit 2000`
Hiermit installieren Sie zunächst einen Netzwerkfilter (Token Bucket Filter, TBF) der für die über das Gerät ifb0 übertragenen Pakete die Rate auf 1024kbit/s begrenzt. Diesem Filter geben Sie den Namen („Handle“) `1:`, so dass wir ihn später wieder ansprechen können.

d. `tc qdisc add dev ifb0 parent 1: handle 10: netem`
Hiermit hängen wir jetzt den eigentlichen Netzwerk-Emulator als Filter ein. Über diesen können wir später die Rate der Paketverluste usw. regulieren. Zunächst wird durch diesen Befehl nur eine Verzögerung von 100 ms eingestellt.

Achtung! Die oben genannten Schritte sind nach einem Neustart ihrer virtuelle Maschine erneut durchzuführen! Wenn Sie sich den Aufwand dafür sparen wollen, können Sie sich beispielsweise ein kurzes Shell-Script erstellen, was diese Schritte durchführt.

Um die Übertragungsrate von TCP auf einfache Art und Weise zu testen, nutzen wir wieder den Befehl „`wget`“. Als Kriterium dient uns die von „`wget`“ beim Aufruf „`wget http://mmix.cs.hm.edu/bin/optmmix-2011-5-6.tgz`“ gemeldete Übertragungsrate.

Führen Sie jetzt folgende Tests durch und notieren Sie jeweils die von `wget` angezeigte Empfangsrate:

a) Bisher haben Sie noch keine Verzögerung, Paketfehler oder ähnliches konfiguriert. Welche Empfangsrate bekommen Sie beim Abruf von `http://mmix.cs.hm.edu/bin/optmmix-2011-5-6.tgz` mit `wget`?

b) Verzögern Sie jetzt alle eingehenden Pakete um 500 ms. Hierzu konfigurieren Sie die Netzwerkemulation über den Befehl
`tc qdisc change dev ifb0 handle 10: netem delay 500ms`
Prüfen Sie Ihr Setup, indem Sie mittels „`ping`“ das Delay zu `www.hm.edu` messen. Messen Sie dann wiederum die TCP-Empfangsrate mit `wget` .Welche Auswirkungen hat das Delay auf die per TCP erzielte Rate? Können Sie sich das Verhalten erklären?

c) Setzen Sie zunächst die Verzögerung wieder auf den Wert 0 zurück. Setzen Sie nun eine Paketverlustrate von zunächst 2% über den Befehl
`tc qdisc change dev ifb0 handle 10: netem loss 2%`
Welche Auswirkungen beobachten Sie auf die Übertragungsrate? Führen Sie mindestens 5 Messungen durch. Steigern Sie nun die Paketverlustrate auf 5%, 10% und 20%. Was beobachten Sie? Wie erklären Sie das beobachtete Verhalten?

d) Setzen Sie zunächst die Paketverlustrate auf eine Wert von 0 zurück. Stellen Sie danach eine Verzögerung von 200ms ein und limitieren Sie die Übertragungsrate auf 128kbit/s indem Sie mit
```
tc qdisc change dev ifb0 handle 1: tbf rate 128kbit buffer 1600 limit 2000
```
Welche Beobachtungen machen Sie jetzt hinsichtlich der mit TCP erzielten Übertragungsrate?

e) Setzen Sie zunächst die Übertragungsrate wieder auf 1024kbit/s hoch und entfernen Sie die künstliche Verzögerung von 200ms. Fügen Sie nun einen weiteren Paketfilter für **ausgehende Pakete** hinzu, der Pakete mit einer Wahrscheinlichkeit von 10% verwirft:
```
tc qdisc add dev eth0 root netem loss 10%
```
Messen Sie erneut die von TCP erzielte Übertragungsrate und vergleichen Sie sie mit der in Aufgabe c) ermittelten Rate. Stellen Sie einen Unterschied fest und – wenn ja – können sie ihn erklären?

f) Konfigurieren Sie Ihre VM wieder wie in Teilaufgabe c) beschrieben. Zeichnen Sie mit Wireshark ein Szenario auf, in dem ein Segmentverlust auftritt. Erläutern Sie das Verhalten von TCP im Fall von Paketverlusten anhand des von Ihnen aufgezeichneten Netzwerk-Traces.

Geben Sie bei der Abgabe bitte alle für die jeweiligen Teilaufgaben gemessenen Übertragungszeiten sowie Ihre Erläuterungen dazu ab.