University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Implementation einer Mobilen Messplattform zur Messung von IP Performance Metriken

Thomas Finkbeiner
Zürich, Schweiz
Matrikelnummer: 99-701-088

Supervisor: Daniel Dönni
Abgabedatum: 30. Januar, 2014

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Zusammenfassung

Das Angebot an mobilen Datendiensten in der Schweiz ist vielfältig. Für den Kunden ist es jedoch schwierig, das Preis-Leistungs-Verhältnis der Dienste verschiedener Anbieter objektiv zu vergleichen. Der Value-of-Service ist ein Konzept zur Messung des Preis-Leistungs-Verhältnisses von IP basierten Diensten. Er basiert auf der Messung des Quality-of-Service von IP Netzwerken. Im Rahmen dieser Arbeit wurde eine Android App mit den dazugehörigen Server-Komponenten entwickelt, die den QoS von mobilen, IP basierten Netzwerken anhand der von der IP Performance Metrics Group der IETF definierten Metriken misst. Sie soll dazu dienen, eine Datenbasis für die Ermittlung des VoS der verschiedenen Mobildienstanbieter aufzubauen, um das Preis-Leistungs-Verhältnis ihrer Dienste objektiv beurteilen zu können.

Danksagung

Ich möchte mich bei Daniel Dönni und Dr. Thomas Bocek für die Betreuung und Unterstützung bei der Erstellung dieser Arbeit bedanken.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Internet Protokoll (IP) basierte Dienste haben nicht nur Telefonie- oder Fax-Dienste revolutioniert, sie sind zu einer Grundlage aller elektronischen Dienste geworden, unabhängig davon, ob Text-, Video-, Bild- oder Audio-Inhalte vermittelt werden. Der Begriff "Dienst" bezeichnet dabei in der Regel den Austausch von Information durch ein Telekommunikationsmittel zwischen Benutzern, unabhängig davon, ob es sich beim Benutzer um eine Person oder ein technisches System handelt [7].

Unter "Dienstqualität" (Quality of Service QoS) wird in der Telekommunikation gemeinhin verstanden, inwieweit ein Dienst die Erwartungen des Benutzers erfüllt [5]. Die Bewertung allerdings erfolgt bei Benutzern und Dienstleistern nach unterschiedlichen Kriterien. Die Definition und Verwendung des Begriffs QoS durch die International Telecommunications Union (ITU), das European Telecommunications Standards Institute (ETSI) und die Internet Engineering Task Force (IETF) kann wie in Abbildung 1.1 verglichen und in ein allgemeines Modell eingebettet werden [5].

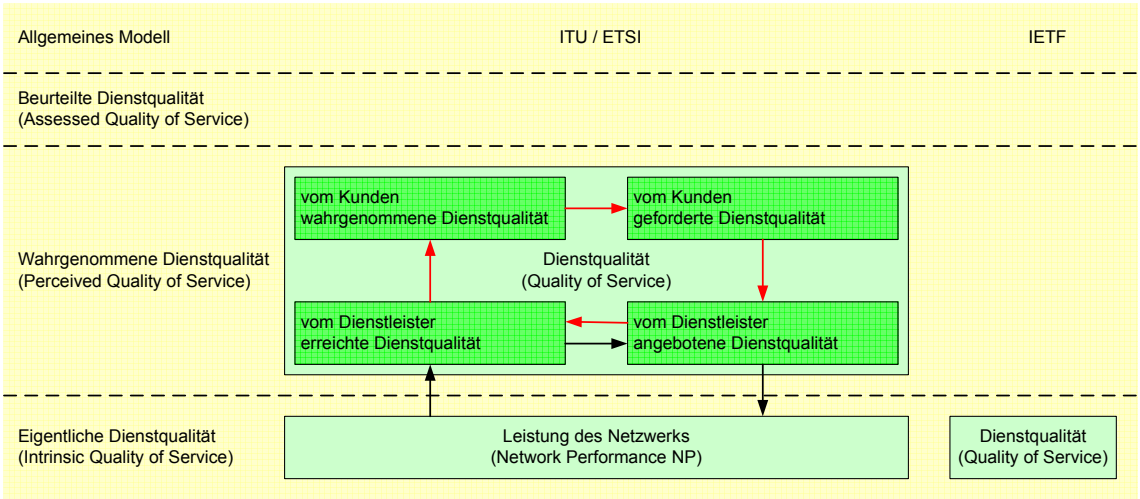

Abbildung 1.1: Ein allgemeines Modell der Dienstqualität und der Ansatz von ITU/ETSI und IETF [5]

Die ”eigentliche Dienstqualität” (intrinsic QoS) bezeichnet die technische Leistung eines Netzwerks. Die geforderte Qualität wird unter anderem durch eine geeignete Wahl von Protokollen und Massnahmen der Qualitätssicherung erreicht. Eine Beurteilung erfolgt durch den Vergleich der gemessenen technischen Werte mit der Spezifikation.

Die ”wahrgenommene Dienstqualität” (perceived QoS) beschreibt die Wahrnehmung der Qualität des Dienstes durch den Kunden. Die Analyse des Begriffs durch die ITU/ETSI zeigt die Problematik: Die durch den Kunden vom Anbieter geforderte Servicequalität begründet sich (auch) aus der wahrgenommenen Qualität der erbrachten Dienstleistung. Die vom Dienstleister effektiv erreichte Qualität kann von der angebotenen abweichen und wird vom Kunden möglicherweise verändert wahrgenommen. Die von einem Dienstleister angebotene Servicequalität muss demzufolge sowohl die eigentliche Dienstqualität als auch die durch den Kunden wahrgenommenen nicht-technischen Qualitätsmerkmale einschliessen.

Die ”beurteilte Dienstqualität” (assessed QoS) wird wirksam, wenn der Kunde über die weitere Nutzung des Dienstes entscheidet. Zusätzlich zur eigentlichen und wahrgenommenen Dienstqualität werden dabei zum Beispiel der Preis oder die Unterstützung des Anbieters bei Störungen in die Entscheidung miteinbezogen. Weder ITU noch ETSI berücksichtigen dies.

Das Konzept des Value-of-Service (VoS) [3], welches Metriken für die Bestimmung des Preis-Leistungs-Verhältnisses von Diensten in IP-Netzwerken definiert, ist ein Element der beurteilten Dienstqualität. Es setzt für jeden Service den eigentlichen QoS in Beziehung zu einem normalisierten Preis. Voraussetzung für die Bestimmung des VoS ist eine Methode zur Messung des eigentlichen QoS für IP Netzwerke. Die wichtigsten Metriken zur Messung des eigentlichen QoS von IP Netzwerken wurden von der IP Performance Metrics (IPPM) Working Group [12] der IETF standardisiert. Bis anhin war jedoch keine Messapplikation für mobile Endgeräte verfügbar, welche diese Metriken tatsächlich misst. Die Entwicklung einer solchen Applikation ist der Beitrag dieser Arbeit.

Implementiert wurde eine Mobile Messplattform für Android-basierte [1] mobile Endgeräte welche aus drei Komponenten besteht. Die erste Komponente ist eine Applikation für die Android Plattform, welche es dem Benutzer erlaubt, Messungen von Metriken zu planen, zu konfigurieren und auszuführen. Die zweite Komponente ist ein sogenannter Reflektor, welche die Messpakete der Android App entgegen nimmt und falls nötig reflektiert. Komplettiert wird das System durch eine dritte Komponente, dem Kollektor, welcher die Messdaten aller Benutzer der Android App sammelt und in einer Datenbank ablegt. Zweck des Systems ist der Aufbau einer Datenbasis, die es ermöglicht, die Dienstleistungen der verschiedenen (Schweizer) Mobildienstleister mit Hilfe des VoS-Konzeptes zu vergleichen.

Kapitel 2

Verwandte Arbeiten

Für die Messung des eigentlichen QoS von Internet Datendiensten, welche mittels Transport Layer Protokollen wie UDP oder TCP über IP übertragen werden, hat die seit 1997 bestehende IPPM Working Group [12] der IETF eine Reihe von Standard Metriken entwickelt und diese in Form von Request for Comments (RFCs) publiziert. Diese Metriken sind so entworfen, dass sie unverfälschte quantitative Leistungsmessungen dieser Dienste ermöglichen. Die Messung dieser Metriken, beziehungsweise die Generierung von Rohdaten zur späteren Auswertung und Verwendung zur Bestimmung des VoS der Dienste verschiedener Telekom Anbieter, ist die Aufgabe der im Rahmen dieser Arbeit entwickelten Mobilen Messplattform. Diese Metriken bilden die Grundlage dieser Arbeit, deshalb werden die wichtigsten in der Folge zusammenfassend dargestellt.

2.1 Framework for IP Performance Metrics

Den Rahmen für die Definition dieser Metriken bildet das "Framework for IP Performance Metrics" RFC 2330 [8]. Dieses beschreibt und definiert unter anderem die Kriterien, welche eine IPPM Metrik erfüllen muss. Wichtig für das Verständnis der folgenden Abschnitte ist die Unterscheidung von "Singleton"-, "Sample"- und "Statistics"-Metriken und das Konzept des "Type-P" Paketes.

Eine Singleton-Metrik bezeichnet eine elementare Metrik. Wird beispielsweise der Paketverlust eines Datendienstes gemessen, so entspräche der Wert der Singleton-Metrik des Paketverlustes dem Ergebnis des Sendens eines einzelnen Paketes. Erreicht das Paket den Empfänger, so wäre der Wert der Singleton-Metrik 0 (kein Verlust), andernfalls 1 (Verlust). Eine Sample-Metrik fasst das Ergebnis mehrerer Singleton-Metriken zusammen. Beim Paketverlust entspräche das dem Senden mehrerer Pakete während eines gewissen Zeitintervalls, und das Ergebnis wäre eine Sequenz von Paaren mit der Sendezeit Ti und dem Ergebnis der Singleton-Metrik, also 0 oder 1. Aus einer Sample-Metrik kann dann eine Statistics-Metrik abgeleitet werden. Eine Statistics-Metrik für den Paketverlust wäre beispielsweise das Verhältnis der verlorenen Pakete zu den gesendeten Paketen.

Da der Wert einer gemessen Metrik meist vom Typ der für die Messung verwendeten IP Pakete abhängt, führt RFC 2330 das Konzept des "Paketes vom Typ P" (Type-P Paket) ein. Dies ist eine generische Bezeichnung und spezifiziert keinen bestimmten Pakettyp, dient jedoch dazu, die Abhängigkeit einer Metrik vom gesendeten Pakettyp explizit zu machen. So müssen alle Metriken, deren gemessener Wert abhängig vom Typ der gesendeten Pakete ist, im Namen entweder einen spezifischen Pakettyp oder die Bezeichnung Type-P enthalten. Der Name einer Metrik für den Paketverlust, die vom Typ der gesendeten Pakete abhängt, ist also nicht IP-One-Way-Packet-Loss, sondern Type-P-One-Way-Packet-Loss.

Die Mobile Messplattfrom implementiert die folgenden neun Metriken, welche von der IPPM Working Group als RFCs publiziert sind [11]. Alle dieser Metriken definieren die beiden Parameter "Src" und "Dst", welche für die IP Adresse des sendenden beziehungsweise empfangenden Hosts stehen und in der Folge in diesem Sinne verwendet werden.

2.2 IPPM Metric for Measuring Connectivity

Konnektivität ist die Basis des Internets, deshalb bildet eine Metrik, welche die gegenseitige Erreichbarkeit zweier Hosts misst, die Grundlage aller anderen Metriken. Die Metriken zur Messung der Konnektivität sind in "IPPM Metric for Measuring Connectivity" RFC 2498 [10] standardisiert. Dieses Dokument spezifiziert eine Reihe aufeinander aufbauender Metriken, welche zur Definition der "Type-P1-P2-Interval-Temporal-Connectivity" Metrik führen. Die Einheit dieser Metrik ist ein boolescher Wert, also entweder "wahr" oder "falsch". Nimmt die Metrik den Wert "wahr" an, so heisst das, dass Src ein Paket an Dst sendet, dass Dst dieses Paket erhält und ein Antwortpaket generiert, welches Src in einem bestimmten Zeitintervall erreicht. Das Zeitintervall ist dabei ein frei wählbarer Parameter der Metrik. Eine Diskussion über mögliche Werte dieses Parameter findet sich in [8]. Zu beachten ist bei dieser Metrik, dass sie nicht vorschreibt, dass das gesendete Paket vom gleichen Typ wie das Antwortpaket sein muss. Der Grund dafür ist, dass viele Anwendungen verschiedene Pakettypen für den Sende- und Antwortverkehr verwenden, deshalb lässt die Metrik verschiedene Pakettypen für gesendete und zurückgesendete Pakete zu.

2.3 One-way Delay Metric for IPPM

Die Metriken zur Messung des One-way Delays finden sich in RFC 2679 "A One-way Delay Metrik for IPPM" [13]. Darin wird zuerst die Singleton-Metrik "Type-P-One-way-Delay" definiert. Der Wert der Metrik ist entweder eine reelle Zahl oder eine "undefinierte" Anzahl von Sekunden. Ist der One-way Delay eine reelle Zahl dT, heisst das, dass Src ein Type-P Paket zur Zeit T gesendet hat, und dass Dst dieses Paket zur Zeit T+dT erhalten hat. Der Wert der Metrik ist "undefiniert", falls Src ein Type-P Paket zur Zeit T gesendet hat, dass dieses Paket Dst jedoch nicht erreicht hat. Ein Parameter dieser Metrik ist der so genannte "Loss-Threshold", welcher bestimmt, wie lange auf ein Paket gewartet wird, bis

der Wert der Metrik als "undefiniert" gilt. Die Metrik "Type-P-One-way-Delay-Poisson-Stream" definiert eine Sample-Metrik, deren Singleton-Metriken mittels eines Poisson-Prozesses gemessen werden. Das heisst, das die Zeitintervalle zwischen den Sendezeiten Ti der einzelnen Type-P Pakete exponentialverteilt gewählt werden. Die Einheit dieser Metrik ist eine Sequenz von Paaren bestehend aus der Sendezeit Ti und entweder einer reellen Zahl dTi oder "undefiniert", also dem Wert des Type-P-One-way-Delays, also eine Sequenz der Form:

<T0, dT0>, <T1, dT1>, <T2, undefiniert>, <T3, dT3>, ...

Ausgehend von der Type-P-One-way-Delay-Poisson-Stream Metrik werden in RFC 2679 eine Reihe von Statistiken definiert, wie zum Beispiel die "Type-P-One-way-Delay-Median" Metrik, welche dem Median aller dT Werte in der Sequenz entspricht. Dabei werden Singleton-Metriken mit dem Wert "undefiniert" als "unendlich lange" betrachtet.

2.4 Round-trip Delay Metric for IPPM

Die Round-Trip Delay Metriken sind in RFC 2681 "A Round-trip Delay Metrik for IPPM" [18] spezifiziert. Die Definitionen der Singleton-Metrik "Type-P-Round-trip-Delay" als auch der "Type-P-Round-trip-Delay-Poisson-Stream" Sample-Metrik sind analog zu den gleichnamigen Type-P-One-way Delay Metriken, nur dass bei den Round-Trip Delay Metriken der Wert T+dT der Zeit entspricht, zu der Src das Antwortpaket von Dst erhält. Eine der Statistiken, welche RFC 2681 definiert, ist das "Type-P-Round-trip-Delay-Minimum", welches dem kleinsten Wert dT im Type-P-Round-trip-Delay-Poisson-Stream entspricht.

2.5 One-way Packet Loss Metric for IPPM

Die "Type-P-One-way-Packet-Loss" Singleton-Metrik sowie alle weiteren One-way Packet Loss Metriken sind in RFC 2680 "A One-way Packet Loss Metrik for IPPM" [16] definiert. Der Wert der Type-P-One-way-Packet-Loss Metrik ist entweder 0 (kein Verlust) oder 1 (Verlust). Ein Wert von 0 bedeutet, dass Src zum Zeitpunkt T ein Type-P Paket an Dst sendet, und dass Dst dieses Paket erhält. Wenn das Paket Dst nicht erreicht, so ist der Wert der Metrik 1. Der Type-P-One-way-Packet-Loss steht in engem Bezug zum Type-P-One-way-Delay, da der Wert des Type-P-One-way-Packet-Loss genau dann 1 ist, wenn der Wert des Type-P-One-way-Delay "undefiniert" ist. Auch beim Type-P-One-way-Packet-Loss muss also ein Wert für den Loss-Threshold Parameter bestimmt werden, und der Wert der Metrik ist genau dann 0, wenn das Paket Dst innerhalb von T + Loss-Threshold erreicht. Der "Type-P-One-way-Packet-Loss-Poisson-Stream" ist eine Sample-Metrik, welche die Werte der Singleton-Metriken mittels eines Poisson-Prozesses misst. Das Ergebnis ist auch hier eine Sequenz von Paaren, bestehen aus der Sendezeit Ti und

dem Wert des Type-P-One-way-Packet-Loss. Die Type-P-One-way-Packet-Loss-Poisson-Stream Metrik für die Type-P-One-way-Delay-Poisson-Stream Metrik für die Sequenz aus 2.3 ist demgemäss:

<T0, 0>, <T1, 0>, <T2, 1>, <T3, 0>, ...

Anhand eines Type-P-One-way-Packet-Loss-Poisson-Streams lässt sich die "Type-P-One-way-Packet-Loss-Average" Metrik bestimmen, welche dem Mittelwert aller Type-P-One-Way-Packet-Loss Metriken in der Sequenz mit dem Wert 1 entspricht. Bei hundert gesendeten und sieben verlorenen Paketen entspräche der Wert der Type-P-One-way-Packet-Loss-Average Metrik also 7%.

2.6 Round-Trip Packet Loss Metrics

Der RFC 6673 "Round-trip Packet Loss Metrics" definiert die Singleton-Metrik "Type-P-Round-trip-Loss" [19], welche analog zur Type-P-One-way-Packet-Loss Metrik die Werte 0 (kein Verlust) oder 1 (Verlust) annehmen kann. Der Wert der Metrik ist 0, falls Src zum Zeitpunkt T ein Type-P Paket an Dst sendet, Dst ein Type-P Paket zurücksendet, und dieses Paket Src innerhalb von T + Loss-Threshold erreicht. Die "Type-P-Round-trip-Loss-<Sample>-Ratio" Metrik basiert auf der "Type-P-Round-trip-Loss-<Sample>-Stream" Metrik und bezeichnet das Verhältnis von verlorenen Paketen zu gesendeten Paketen. <Sample> steht dabei für die Methode, mit welcher die Stichprobe der Singleton-Metriken erzeugt wird, und muss durch die gewählte Methode ersetzt werden, also beispielsweise durch "Poisson" oder "Periodic".

2.7 IP Packet Delay Variation Metric for IPPM

Die in RFC 3393 "IP Packet Delay Variation Metric for IPPM" [9] spezifizierte "Type-P-One-way-ipdv" Singleton-Metrik basiert auf der Type-P-One-way-Delay-Poisson-Stream Sample-Metrik. Der Wert dieser Metrik ist entweder eine reelle Zahl oder eine "undefinierte" Anzahl von Sekunden. Der Wert der reellen Zahl kann dabei positiv, negativ oder 0 sein. Zur Ermittlung der Type-P-One-way-ipdv Singleton-Metrik werden aus einem Type-P-One-way-Delay-Poisson-Stream mittels einer Selektionsfunktion F zwei Paare selektiert, für deren Sendezeiten die Bedingung T1 < T2 gilt. Der Wert der Type-P-One-way-ipdv Metrik entspricht dann der Differenz dT2 - dT1=ddT, also der Differenz des One-way Delays. Ist einer der Werte dT1 oder dT2 "undefiniert", so ist auch der Wert der Type-P-One-Way-ipdv Metrik "undefiniert". Die "Type-P-One-way-Delay-ipdv-Poisson-Stream" Sample-Metrik ist eine Sequenz von Tripeln, bestehend aus den Sendezeiten T1 und T2 und einer reellen Zahl ddT oder einer "undefinierten" Anzahl von Sekunden. Von der "Type-P-One-way-Delay-ipdv-Poisson-Stream" Metrik können mehrere Metriken abgeleitet werden, wie beispielsweise die "Type-P-One-way-ipdv-percentile" Metrik, deren Wert ein bestimmtes Perzentil aller IP Delay Variation Werte aus der Stichprobe darstellt.

2.8 One-way Loss Pattern Sample Metrics

Mehrere auf der Type-P-One-way-Packet-Loss-Poisson-Stream Metrik aufbauende Metriken spezifiziert RFC 3357 "One-way Loss Pattern Sample Metrics" [14]. Dazu werden folgende Begriffe eingeführt, die sich auf eine Sequenz von Paketen beziehen, die entweder verloren gegangen oder erfolgreich übertragen wurden, beispielsweise

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	R	R	X	R	R	X	X	X	R	X	R	R	X	X	X

wobei mit R diejenigen Pakete bezeichnet werden, welche erfolgreich übertragen wurden und X für verlorene Pakete steht.

Loss Distance Die "Loss Distance" bezeichnet die Differenz der Sequenznummern zweier verlorener Pakete. Die Loss Distance der Pakete P(6) und P(7) im Beispiel ist also 1, die Loss Distance von Paket P(10) und Paket(P13) beträgt 3.

Loss Period Eine "Loss Period" beginnt immer, wenn das Paket mit Sequenznummer P(i) ein verlorenes Paket darstellt, und das Paket mit Sequenznummer P(i-1) ein empfangenes Paket. Eine Loss Period beginnt also immer mit dem Muster "R X". Das Beispiel enthällt also vier Loss Periods, welche mit den Paketen P(3), P(6), P(10) und P(13) beginnen.

Aufbauend auf diesen Definitionen spezifiziert RFC 3357 die beiden Sample-Metriken "Type-P-One-way-Loss-Distance-Stream" und "Type-P-One-way-Loss-Period-Stream". Der Wert der Type-P-One-way-Loss-Distance-Stream Metrik ist eine Sequenz aus Paaren der Form <loss distance, loss>, wobei "loss" aus der Type-P-One-way-Packet-Loss-Poisson-Stream Metrik abgeleitet wird und die "loss distance" entweder 0 oder eine positive Zahl ist. Falls 'loss" den Wert 1 hat, ist die "loss distance" wie oben beschrieben die Differenz der Sequenznummern der Pakete. Der Wert der Type-P-One-way-Loss-Period-Stream Metrik ist ist eine Sequenz von Paaren der Form <loss period, loss>, wobei die "loss period" für eine ganze Zahl steht, welche bei jeder Loss Period in der Sequenz inkrementiert wird und zu Beginn 0 ist. Bei einem gegebenen Type-P-One-way-Loss-Poisson-Stream der Form

<T1,0>,<T2,1>,<T3,0>,<T4,0>,<T5,1>,<T6,0>,<T7,1>,<T8,0>,<T9,1>,<T10,1>

ist der Wert des Type-P-One-Way-Loss-Distance-Stream (<loss distance, loss>) also

<0,0>,<0,1>,<0,0>,<0,0>,<3,1>,<0,0>,<2,1>,<0,0>,<2,1>,<1,1>

da die Loss Distance für das erste verlorene Paket als 0 definiert ist. Für den Type-P-One-way-Loss-Period-Stream <loss period, loss> ist der Wert

<0,0>,<1,1>,<0,0>,<0,0>,<2,1>,<0,0>,<3,1>,<0,0>,<4,1>,<4,1>

da der Type-P-One-way-Loss-Poisson-Stream vier Loss Perioden enthält.

Alle weiteren in RFC 3357 definierten Metriken werden von diesen zwei Sample-Metriken abgeleitet. Die "Type-P-One-way-Loss-Period-Total" Metrik ist definiert als der maximale Wert des ersten Elements der Paare aus einem Type-P-One-way-Loss-Period-Stream, was im obigen Beispiel dem Wert vier entsprechen würde. Eine weitere Metrik, die "Type-P-One-way-Loss-Period-Lengths", misst die Länge der Loss Perioden in einem Type-P-One-way-Loss-Period-Stream, also wie wie viele aufeinanderfolgende verlorene Pakete die jeweiligen Loss Perioden enthalten, und die "Type-P-One-way-Inter-Loss-Period-Lengths" Metrik sagt aus, wie viele Pakete erfolgreich zwischen zwei Loss Perioden übertragen werden konnten.

2.9 One-way Packet Duplication Metric

Die One-way Packet Duplication Metriken sind RFC 5560 "A One-way Packet Duplication Metrik" [15] spezifiziert. Der Wert der "Type-P-One-way-Packet-Duplication" Metrik ist eine positive Zahl, welche die Anzahl von zusätzlichen, identischen Paketen bezeichnet, welche Dst erreichen, nachdem Src ein Type-P Paket gesendet hat. Erhält Dst nur ein Paket, so ist der Wert der Metrik 0. Falls kein Paket bei Dst eintreffen, ist der Wert der Metrik undefiniert. Die "Type-P-One-way-Packet-Duplication-Fraction" Metrik drückt den Anteil von zusätzlichen Paketen in einer Paketsequenz aus, welche bei Dst eintreffen, also das Verhältnis von erhaltenen Paketen zu gesendeten Paketen minus 1, wobei verlorene Pakete nicht zu den gesendeten Paketen gezählt werden. Die "Type-P-One-way-Replicated-Packet-Rate" dagegen gibt den Anteil an Paketen in einer Paketsequenz an, welche einmal oder mehrere male dupliziert wurden, also die Anzahl von duplizierten Paketen zur Anzahl der gesendeten Pakete, wobei auch hier dir verlorenen Pakete von den gesendeten Paketen subtrahiert werden. Sendet Src also die Paketsequenz (1, 2, 3, 4) und Dst erhält (1, 1, 1, 2, 3, 3, 3, 4), so entspricht die Type-P-One-way-Packet-Duplication-Fraction dem Wert 1 oder 100%, und die Type-P-One-way-Replicated-Packet-Rate dem Wert 0,5 oder 50%.

2.10 Packet Reordering Metrics

Die in RFC 4737 "Packet Reordering Metrics" [17] beschriebene Singleton-Metrik "Type-P-Reordered" kann am besten anhand eines Algorithmus dargestellt werden. Dabei wird von einer Paketsequenz ausgegangen, deren Pakete streng monoton steigende Sequenznummern aufweisen. Ausserdem wird eine Variable "nextExpected" benötigt, welche die nächste erwartete Sequenznummer speichert und zu beginn mit 0 initialisiert wird. Der Wert der Type-P-Reorderd Singleton-Metrik ist entweder wahr oder falsch. Wenn Src ein Type-P Paket mit Sequenznummer s an Dst sendet, und dieses bei Dst eintrifft, so wird der Wert der Type-P-Reordered Metrik mit dem Algorithmus 1 bestimmt. Die Definition der Sample-Metrik "Type-P-Reordered-Ratio-Stream" wird anhand dieser Type-P-Reorderd Metrik definiert. Sie bezeichnet das Verhältnis der Anzahl von Paketen mit dem Type-P-Reordered Wert "wahr" zur Anzahl der gesendeten Paketen minus Duplikate.

Algorithm 1 Type-P-Reordered

$nextExpected \leftarrow 0$
if $s >= nextExpected$ **then**
  $nextExpected \leftarrow s + 1$
  $Type - P - Reordered \leftarrow false$
else
  $Type - P - Reordered \leftarrow true$
end if

Kapitel 3

Systemarchitektur

Im Rahmen der vorliegenden Arbeit sollte eine Applikation entwickelt werden, welche es erlaubt, Messreihen für IP Performance Metriken durchzuführen und damit Messwerte für die in Kapitel 2 vorgestellten Metriken zu ermitteln. Da die ermittelten Messwerte dazu dienen sollen, mittels des VoS mobile Datendienste von Telekom-Anbietern zu vergleichen, war in der Aufgabenstellung die Android Plattform als Zielplattform vorgegeben. Eine weitere benötigte Komponente neben einer Android App, mit welcher die Messungen initiiert und Messpakete generiert werden können, ist ein Server, welcher die gesendeten Pakete entgegen nimmt und gegebenenfalls reflektiert. Diese Komponente des Systems, in der Folge als Reflektor bezeichnet, muss den Datenverkehr zusätzlich auch aufzeichnen, da zur Auswertung der Messungen sämtliche Messpakete benötigt werden, welche auf dem Client und auf dem Reflektor ein- und ausgegangen sind. Um die Messdaten persistent speichern zu können, wird als dritte Komponente ein weiterer Server benötigt, der Kollektor, welcher die Messdaten entgegennimmt und in einer Datenbank ablegt. Aus dieser Systembeschreibung lassen sich damit folgende für die Architektur relevanten Anforderungen ableiten:

- Die Messdaten, welche auf dem Client (App) und dem Reflektor anfallen, müssen nach Abschluss der Messung zusammengeführt werden.
- Aus den zusammengeführten Messdaten müssen die Werte der Metriken berechnet werden und dem Client zur Verfügung stehen, damit diese dem Benutzer angezeigt werden können.
- Die zusammengeführten Messdaten müssen dem Kollektor zur Speicherung zur Verfügung gestellt werden.

Damit ergaben sich für die Architektur drei mögliche Varianten, welche in Betracht gezogen wurden. Diese werden nachfolgend skizziert.

In Variante 1 senden sowohl der Client als auch der Reflektor nach Abschluss der Messung ihre gesammelten Messpakete an den Kollektor. Dieser berechnet bei Erhalt der Daten die Messresultate und legt die Messdaten in der Datenbank ab. Um dem Benutzer die

Messresultate präsentieren zu können, muss der Client den Kollektor anfragen, ob dieser die Messdaten des Reflektors bereits erhalten und die Messresultate berechnet hat. Ist dies der Fall, so lädt der Client die Messresultate vom Kollektor, um sie dem Benutzer anzeigen zu können, andernfalls muss die Anfrage periodisch wiederholt werden, bis die Resultate bereitstehen. Dieses Szenario ist in Abbildung 3.1 dargestellt.

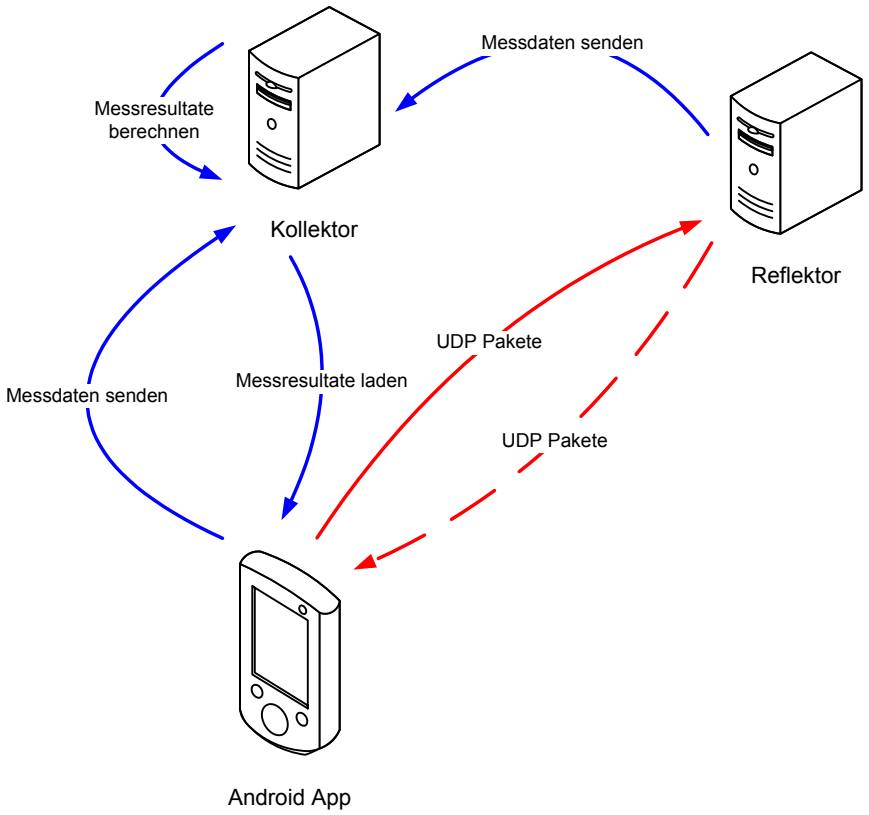

Abbildung 3.1: Systemarchitektur Variante 1

Demgegenüber sendet in Variante 2 der Client seine Messdaten nicht an den Kollektor, sondern an den Reflektor. Dieser berechnet daraufhin die Messresultate und sendet sowohl die Messdaten des Clients als auch die von ihm aufgezeichneten Pakete an den Kollektor. In dieser Variante fragt der Client den Reflektor an, ob dieser die Messresultate berechnet hat, und lädt diese nach Abschluss der Berechnung, um sie dem Benutzer zugänglich zu machen. Abbildung 3.2 zeigt dieses Szenario.

In Variante 3 lädt der Client die Messdaten des Reflektors nach Beendigung der Messung und sendet die zusammengeführten Daten wie in Abbildung 3.3 dargestellt an den Kollektor. Die Berechnung der Messresultate findet auf dem Client statt, der sie anschliessend dem Benutzer präsentieren kann.

Die drei Varianten wurden unter den Gesichtspunkten Verfügbarkeit von Systemressourcen, Kopplung der Komponenten und Komplexität der Implementierung evaluiert. Bei der Verfügbarkeit der Systemressourcen stellte sich die Frage, ob das Berechnen der Resultate

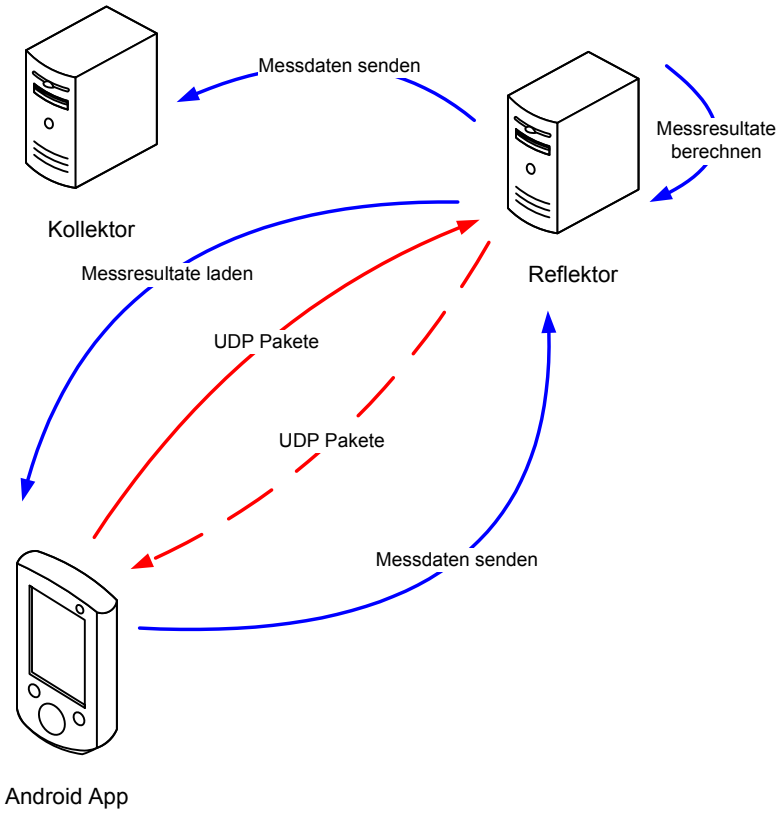

Abbildung 3.2: Systemarchitektur Variante 2

auf dem mobilen Client hinsichtlich Rechenleistung und Speicherbedarf machbar ist. Tests zu Beginn der Arbeit haben gezeigt, dass die benötigte Rechenleistung zur Berechnung der Resultate klein ist, und dass der Speicherbedarf im Kilobyte-Bereich liegt und somit handhabbar ist. Damit wurde dieses Kriterium für die weitere Evaluation der Architektur nicht mehr berücksichtigt.

Vergleicht man die Kopplung der Komponenten in den drei Varianten fällt auf, dass in Variante 1 sämtliche Komponenten miteinander kommunizieren und gekoppelt sind. In Variante 2 fällt die Kopplung vom Client zum Kollektor weg, in Variante 3 die Kopplung von Reflektor und Kollektor. Wenn eine lose Kopplung angestrebt wird, bieten also die Varianten 2 und 3 Vorteile.

Hinsichtlich der Komplexität der Implementierung wurde das Abrufen der Messdaten unmittelbar nach Beendigung der Messung in Variante 3 als einfacher und weniger aufwändig zu implementieren eingestuft als das möglicherweise mehrmalige Verbinden zu Kollektor oder Reflektor in den Varianten 1 und 2, da die Messdaten nach Beendigung der Messung mit Sicherheit zur Verfügung stehen.

Unter diesen Gesichtspunkten ist also Variante 3 die am besten geeignete Architektur für das System. Da sie zudem die Möglichkeit bietet, die Kollektor- und Reflektor-Komponente nicht nur unabhängig voneinander zu entwickeln sondern später auch zu

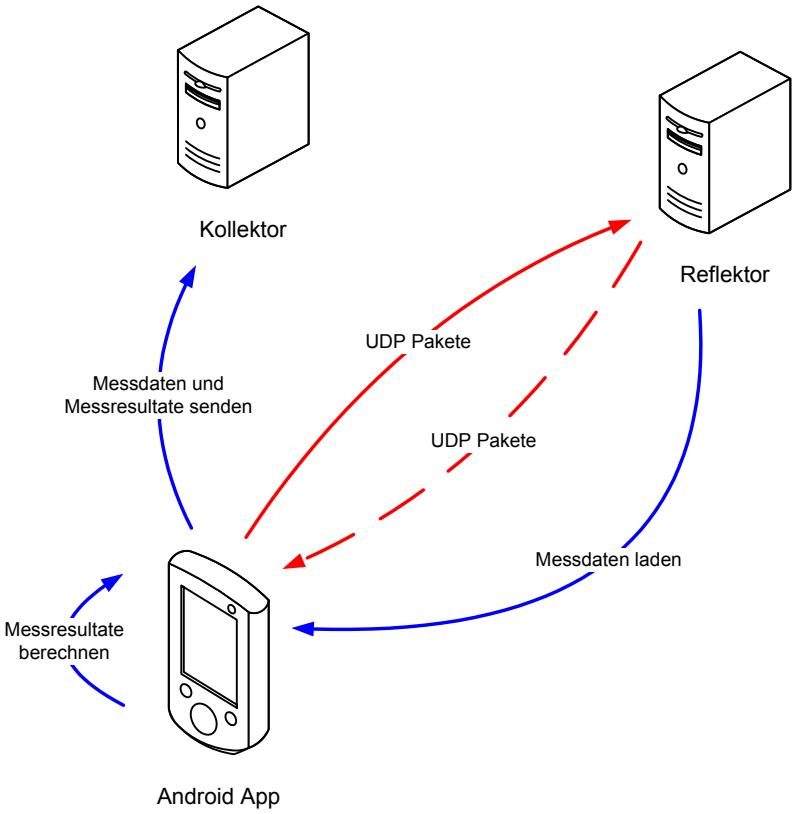

Abbildung 3.3: Systemarchitektur Variante 3

betreiben, wurde das System nach Variante 3 implementiert.

Kapitel 4

Design und Implementierung

Die Entwicklung der in Kapitel 3 vorgestellten Mobilen Messplattform sollte in drei Schritten erfolgen. Zuerst sollte der domänenspezifische Teil der Applikation geschrieben werden, der es erlaubt, die Messungen von den in Kapitel 2 beschriebenen Metriken zu implementieren. Darauf aufbauend sollten in einem zweiten Schritt die IPPM Messungen entwickelt und in einem dritten Schritt die Android App implementiert werden. Die Ziele des Designs waren:

- Trennung des UI und des Domain Layers, damit die API unabhängig von der eingesetzten UI-Technologie verwendet werden kann und nicht an Android gebunden ist.
- Abstrahierung von den Klassen im java.net Package, so dass bei der Implementierung von IPPM Messungen eine einfache API zur Verfügung steht und insbesondere keine Ausnahmebehandlung nötig ist.
- Die (nachträgliche) Implementation von auf UDP basierenden Messungen für IPPM Metriken soll einfach möglich sein.

Mit diesen Vorgaben wurde ein erstes Design erstellt und damit begonnen, das System zu implementieren. Leider hat sich dabei die oben skizzierte Vorgehensweise als nicht optimal erwiesen, denn nach der Entwicklung eines Grossteils des Programmcodes des Domain Layers scheiterte der Versuch, diesen auf der Android Plattform einzusetzen. Die Gründe dafür waren einerseits die Verwendung von Fremdbibliotheken, welche für Android (zur Zeit) nicht unterstützt werden, wie etwa die Java-Bean Validation Implementierung von Hibernate [6]. Andererseits wurden gewisse Anforderungen und Besonderheiten von Android aus Mangel an Erfahrung mit der Plattform im Entwurf nicht berücksichtigt, wie etwa, dass beim Wechseln der Ansichten (in Android "Activity" genannt) alle Objekte zerstört werden, welche nicht in statischen Variablen gehalten werden. Dies hatte zur Folge, dass ein Teil des Designs und der Implementierung überarbeitet und angepasst werden musste, um diese auf Android lauffähig zu machen. Die wichtigsten Klassen und Interfaces des resultierenden und nun implementierten Entwurfs werden in den folgenden Abschnitten dargestellt.

4.1 Implementation von IPPM Messungen

Um die Messung einer IPPM Metrik zu implementieren, werden die in Abbildung 4.1 dargestellten Klassen und Interfaces verwendet. Eine Messung wird dabei mit Hilfe des Interfaces `Measurement` implementiert. Wie dem Klassendiagramm zu entnehmen ist, besitzt dieses Interface eine Methode `execute(MeasurementSocket)`. Dies ist die erste Methode, die von der Applikation aufgerufen wird, wenn die Messung ausgeführt wird. Beim Start der Messung erhalten wir also von der Applikation eine Referenz auf einen `MeasurementSocket`. Ein `MeasurementSocket` besitzt nur eine Methode, nämlich die Methode `send(MeasurementPacket)`. Betrachtet man sich die Klassendefinition der Klasse `MeasurementPacket`, so sieht man, dass dies eine konkrete Klasse ist und wir ein `MeasurementPacket` instanzieren können. Somit haben wir die Möglichkeit, ein `MeasurementPacket` an den Reflektor zu senden und mit der Messung zu beginnen. Das Interface `Measurement` definiert des Weitern noch die beiden Methoden `receive(MeasurementPacket)` und `receive(MeasurementData)`. Die erste dieser beiden Methoden wird von der Applikation jedes mal aufgerufen, wenn ein Antwortpaket vom Reflektor eintrifft. Im Gegensatz dazu wird die Methode `receive(MeasurementData)` nur einmal aufgerufen, nämlich wenn die Messung beendet ist und die Messdaten vom Reflektor abgerufen wurden. Das `MeasurementData` Objekt enthält sämtliche Messpakete, die auf dem Client und auf dem Reflektor aufgezeichnet wurden, und somit können hier die Resultate der Messungen ermittelt werden.

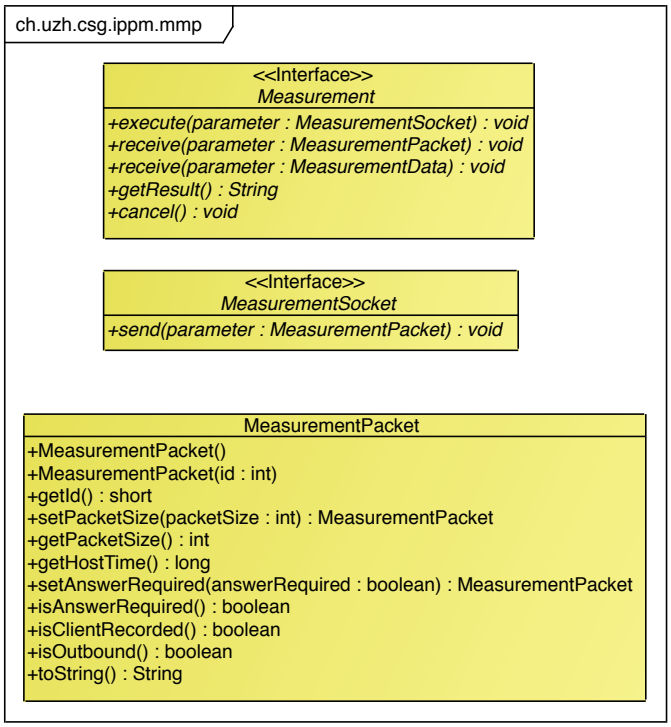

Abbildung 4.1: Benötigte Klassen und Interfaces zur Implementierung einer IPPM Messung

Wie oben beschrieben war es eines der Ziele des Designs, eine Abstraktion für die Klassen aus dem java.net Package zu finden. Die eben vorgestellten Klassen und Interfaces realisieren diese Abstraktion für die beiden Klassen `java.net.DatagramPacket` und `java.net.DatagramSocket`.

4.2 Schnittstelle UI Layer - Domain Layer

Die in Abbildung 4.2 gezeigten Klassen bilden die Schnittstelle vom UI Layer zum Domain Layer. Sie sind die ersten Objekte unterhalb des UI Layers, welche Anfragen des UI Layers erhalten und werden in [20] als "Session Controller" oder "Use-Case Controller" bezeichnet. Eine Ableitung der abstrakten Klasse `MeasurementConfiguration` wird immer zusammen mit einer Messung implementiert und ist verantwortlich für die Erzeugung der konkreten `Measurement` Objekte. Gleichzeitig wird sie vom UI Layer für die Konfiguration von Messreihen für die von ihr instanzierten Messungen verwendet.

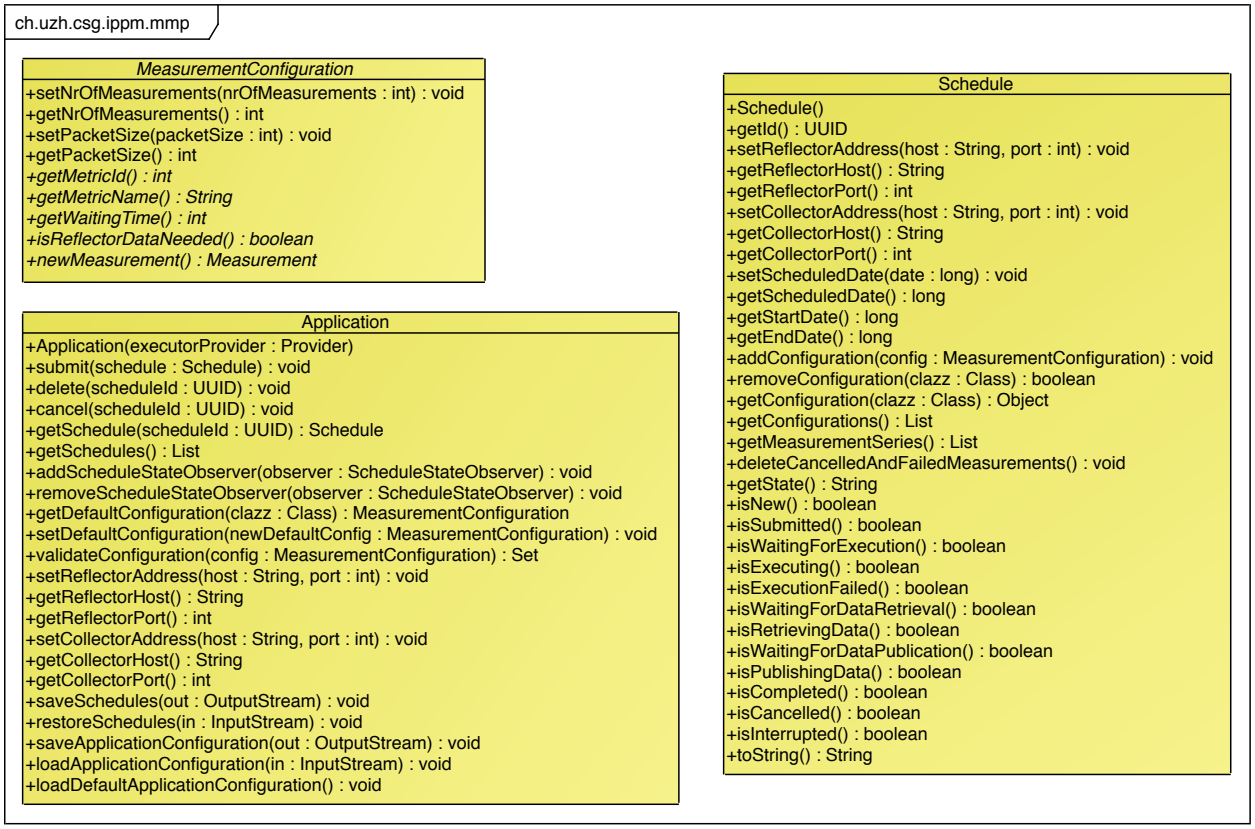

Abbildung 4.2: Session Controller Klassen des Domain Layers

Die Klasse `Schedule` realisiert die in der Aufgabenstellung formulierte Anforderung, wonach die Benutzer der Applikation die Möglichkeit haben sollen, Zeitpläne zu erstellen und

zu definieren, welche Metriken zum geplanten Zeitpunkt gemessen werden sollen. Diese Anforderung wird durch die Methoden `addConfiguration(MeasurementConfiguration)` und `setScheduledDate(long)` realisiert. Daneben dient ein `Schedule` auch als Datenbehälter für alle Messdaten und Messpakete, welche die mit dem `Schedule` Objekt geplanten Messungen bei der Ausführung generieren. Um die in einem `Schedule` definierten Messungen tatsächlich auszuführen, wird die Klasse `Application` benötigt, welche die Methode `submit(Schedule)` zur Verfügung stellt. Sobald diese Methode aufgerufen wird, plant die Applikation die Ausführung der Messungen gemäss der im `Schedule` festgelegten Zeit.

4.3 Serialisierung

In einer ersten Version wurde die Serialisierung der Datenobjekte zur Speicherung auf dem Client und zur Übertragung der Daten über das Netzwerk mit Hilfe der Java Standard-Serialisierung implementiert. Da diese aber Änderungen an der Implementation der serialisierbaren Klassen kaum zulässt und die korrekte Implementation der Standard-Serialisierung nicht trivial ist [2], wurde ein anderes Serialisierungs-Framework evaluiert. Gewählt wurde das Kryo-Serialization-Framework [4], da es eine gewisse Vorwärts- und Rückwärtkompatibilität bietet. Das heisst, dass bereits serialisierte Objekte immer noch gelesen werden können, wenn Felder zu einer Klasse hinzugefügt oder von einer Klasse entfernt werden, nur das Ändern des Typs des Feldes wird nicht unterstützt. Im Vergleich zur Standard-Serialisierung bietet dieses Framework aber damit erheblich mehr Flexibilität.

Kapitel 5

Zusammenfassung und Schlussfolgerungen

Im Rahmen dieser Arbeit wurde eine mobile Messplattform entwickelt, die es erlaubt, Messungen und Messreihen für die in Kapitel 2 zusammengefassten IP Performance Metriken durchzuführen. Entwickelt wurde dafür eine Android App, welche eine grafische Benutzeroberfläche zur Verfügung stellt, die es dem Benutzer ermöglicht, Messreihen zu planen, zu konfigurieren und auszuführen. Der Benutzer hat ebenfalls die Möglichkeit, sich die ermittelten Messwerte anzeigen zu lassen. Als zweite Komponente wurde ein Reflektor implementiert, welcher die von der App generierten und gesendeten Messpakete entgegennimmt, gegebenenfalls reflektiert, und die ein- und ausgehenden Messpakete speichert, damit die Messdaten von der App nach Beendigung der Messungen geladen werden können. Die dritte implementierte Komponente des Systems, der Kollektor, sammelt die Messdaten der Messungen aller Clients und legt sie in einer Datenbank ab, damit diese zur späteren Analyse zur Verfügung stehen.

Die Evaluation der Architekturvarianten führte zu einer Architektur mit zwei unabhängigen Server-Komponenten, bei der die App-Komponente alle Datentransfers initiiert. Das Design der Applikation wurde so gewählt, dass es möglich sein sollte, Messungen für weitere Metriken einfach zu implementieren. Die beiden Server-Komponenten sowie auch die Android App wurden in Java geschrieben. Zur Speicherung der Messdaten wird die Java Persistence API mit der Referenz-Implementation EclipseLink˜ eingesetzt, und zur Serialisierung der Objekte wird das Kryo-Serialization-Framework [4] verwendet.

Die grösste Herausforderung bei der Entwicklung der mobilen Messplattform war die Implementierung der Nebenläufigkeit. Da auf viele Klassen des Domain Layers parallel zugegriffen wird oder aber die Klassen selbst aktive Objekte sind, mussten diese synchronisiert werden. Schwierig war dabei nicht nur die Implementation, sondern auch das Testen dieser Klassen.

Ein Problem bei Messungen von IPPM Metriken, welche auf dem Type-P-One-way-Delay basieren, ist immer die Synchronisation der Uhren von Quell- und Zielhost. Dieses Problem wurde im Rahmen dieser Arbeit insofern berücksichtigt, als dass alle Messungen von Metriken, welche auf dem Type-P-One-way-Delay aufbauen, mit einem Parameter

zur Eingabe eines Schätzwertes für die Uhrendifferenz konfiguriert werden können. Um aber tatsächlich mit synchronen Uhren arbeiten zu können, bedürfte es vermutlich einer anderen Infrastruktur. Trotzdem sollte es möglich sein, mit Hilfe der Werte des Type-P-One-way-Delay und des Type-P-Round-trip-Delay eine obere Schranke für die Differenz der Uhren zu schätzen, so dass für alle implementierten Metriken ausser dem Type-P-One-way-Delay aussagekräftige Werte ermittelt werden können. Somit sollte es also möglich sein, mit Hilfe der entwickelten Messplattform den Quality-of-Service Parameter des Value-of-Service zu bestimmen.

Literaturverzeichnis

[1] Android.com: "Android"; http://www.android.com. Last visited: 21.01.2014.

[2] J. Bloch: "Effective Java"; Addison-Wesley, 2nd edition, 2008.

[3] D. Dönni: "Value-of-Service (VoS)"; 1st Workshop on Large Scale Network Measurements (31st NMRG meeting) Zürich, Switzerland, October 2013.

[4] Esoterik Software: "EsotericSoftware/kryo · GitHub"; https://github.com/EsotericSoftware/kryo. Last visited: 21.01.2014.

[5] J. Gozdecki, A. Jajszczyk and R. Stankiewicz: "Quality of Service Terminology in IP Networks"; Communications Magazine IEEE, Vol. 41, Issue 3, 2003. URL: http://dx.doi.org/10.1109/MCOM.2003.1186560. Last visited: 21.01.2014.

[6] Hibernate.org: "Hibernate Validator - Hibernate Validator"; http://hibernate.org/validator. Last visited: 21.01.2014.

[7] International Telecommunication Union: "Support of IP-based services using IP transfer capabilities"; http://www.itu.int/rec/T-REC-Y.1241-200103-I/en. Last visited: 21.01.2014.

[8] Internet Engineering Task Force: "Framework for IP Performance Metrics"; http://tools.ietf.org/html/rfc2330. Last visited: 21.01.2014.

[9] Internet Engineering Task Force: "IP Packet Delay Variation Metric for IPPM"; http://tools.ietf.org/html/rfc3393. Last visited: 21.01.2014.

[10] Internet Engineering Task Force: "IPPM Metric for Measuring Connectivity"; http://tools.ietf.org/html/rfc2498. Last visited: 21.01.2014.

[11] Internet Engineering Task Force: "IIP Performance Metrics (ippm) - Documents"; http://datatracker.ietf.org/wg/ippm. Last visited: 21.01.2014.

[12] Internet Engineering Task Force: "IP Performance Metrics (ippm) - History"; http://datatracker.ietf.org/wg/ippm/history. Last visited: 21.01.2014.

[13] Internet Engineering Task Force: "One-way Delay Metric for IPPM"; http://tools.ietf.org/html/rfc2679. Last visited: 21.01.2014.

[14] Internet Engineering Task Force: "One-way Loss Pattern Sample Metrics"; http://tools.ietf.org/html/rfc3357. Last visited: 21.01.2014.

[15] Internet Engineering Task Force: ”One-way Packet Duplication Metric”; http://tools.ietf.org/html/rfc5560. Last visited: 21.01.2014.

[16] Internet Engineering Task Force: ”One-way Packet Loss Metric for IPPM”; http://tools.ietf.org/html/rfc2680. Last visited: 21.01.2014.

[17] Internet Engineering Task Force: ”Packet Reordering Metrics”; http://tools.ietf.org/html/rfc4737. Last visited: 21.01.2014.

[18] Internet Engineering Task Force: ”Round-trip Delay Metric for IPPM”; http://tools.ietf.org/html/rfc2681. Last visited: 21.01.2014.

[19] Internet Engineering Task Force: ”Round-Trip Packet Loss Metrics”; http://tools.ietf.org/html/rfc6673. Last visited: 21.01.2014.

[20] C. Larman: ”Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and Iterative Development”; Prentice Hall, 3rd edition, 2004.

Abkürzungen

TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Glossar

Host Ein Rechner, welcher die Fähigkeit hat, mittels Internet Protokollen zu kommunizieren

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Benutzungsleitfaden

A.1 Android App

Die Android App ist auf Google Play erhältlich. Abbildung A.1 zeigt die verfügbaren Messparameter:

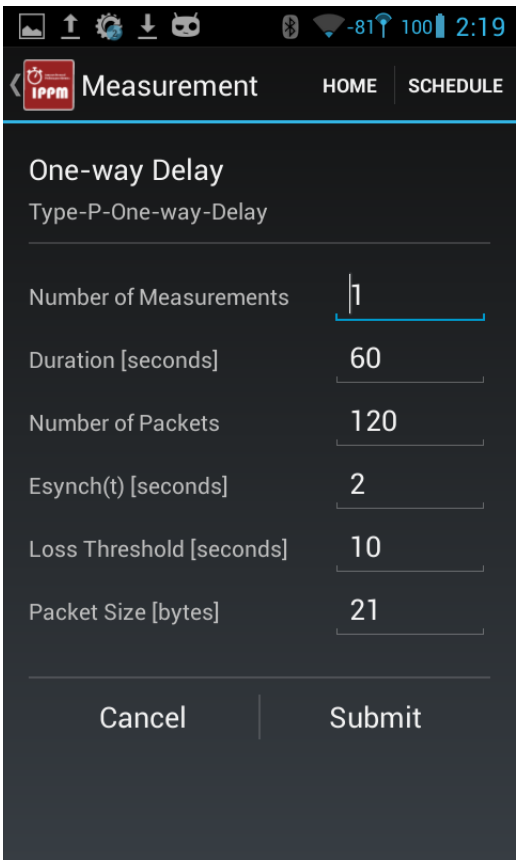

Abbildung A.1: Verfügbare Messparameter

Number of Measurements Hier kann die Anzahl der Messungen angegeben werden, die durchgeführt werden sollen.

Duration Hier wird die Dauer der Messung in Sekunden angegeben.

Number of Packets Spezifiziert die Anzahl der Pakete, welche in der angegebenen Dauer gesendet werden.

Esynch(t) Hier kann eine Schätzung für die Asynchronität der Uhren in Sekunden angegeben werden.

Loss-Threshold Gibt die Zeit in Sekunden an, nach der ein Paket als verloren gewertet wird.

Packet Size Hier kann die Grösse in Bytes der Messpakete spezifiert werden.

A.2 Reflektor und Kollektor

Der Reflektor bzw. Kollektor kann mit folgender Eingabe gestartet werden:

```
java -jar mmp-reflector-1.0.0.jar [ -help | -h <addresse> | -p <port> | -c
<config-file>]
```

```
java -jar mmp-collector-1.0.0.jar [ -help | -h <addresse> | -p <port> | -c
<config-file>]
```

Standardmässig wird dabei ein Konfigurationsfile in dem Ordner `.mmp` im Heimverzeichnis des Benutzers angelegt. Dieses Verhalten kann durch setzen der System-Property `reflector.home` bzw. `collector.home` beim Starten überschrieben werden.