PR Rechnernetze und Netzwerkprogrammierung 621.801/621.802	WS 2024/2025
Institut für Informationstechnologie (ITEC)	Rainer/Timmerer
Übungsblatt 04	

Ü 4.1 OMNeT++ installieren & TicToc-Tutorial

Installieren Sie den Netzwerksimulator OMNeT++ 6.1.0 (siehe http://www.omnetpp.org) und machen Sie sich mit der Software vertraut. Hinweis: Dokumentieren Sie alle Schritte, falls Sie auf Probleme stoßen. (Hinweis: Beachten Sie die Schritte in der Datei INSTALL.md.)

Arbeiten Sie sich in einem ersten Schritt durch das TicToc-Tutorial (siehe https://docs.omnetpp.org/tutorials/tictoc/) und erstellen bzw. starten Sie die entsprechenden Dateien. Modifizieren Sie dieses Tutorial, um sich mit OMNeT++ und dessen Eigenheiten vertraut zu machen. Beispielsweise können Sie die Netzwerktopologie verändern oder das Nachrichtenformat, um eigene Felder zu erweitern, usw.

Gehen Sie vor allem auf folgende Themen bzw. Fragestellungen ein und verfassen Sie einen Erfahrungsbericht inkl. Screenshots:

- Wie wird eine Topologie erstellt und dessen Funktionalität realisiert?
- Kompilieren und Starten einer Simulation.
- Hinzufügen von graphischen Elementen, Ausgaben zur Fehlersuche, Zustandsvariablen und (zufälligen) Parametern.
- Vererbung, Verzögerung, Zeitüberschreitungen und dessen Aufhebungen.
- Netzwerktopologien mit mehr als zwei Knoten.
- Wie kann ein eigenes Nachrichtenformat definiert und verwendet werden?
- Hinzufügen von Statistiken zur anschließenden Auswertung und Visualisierung.

Ü 4.2 OMNeT++ .ned-Datei erstellen

Erstellen sie die folgende Topologie `simpleNet.ned` (siehe Abbildung 1) mit dem graphischen Editor.

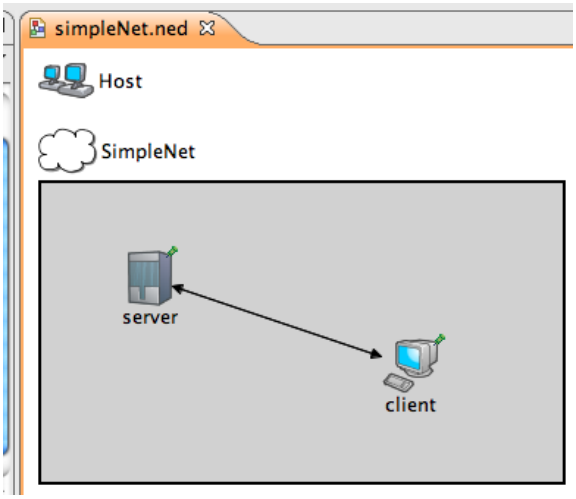

Abbildung 1: Beispiel-Netzwerktopologie.

Die Module `server` und `client` sollen als einfaches Modul dargestellt. Der darunterliegende Kanal (`channel`) soll eine Bandbreite von 10 Mbit/s und eine Verzögerung (`delay`) von 5 Millisekunden haben. Erstellen Sie eine C++-Datei mit den notwendigen Klassendefinitionen und leeren Methodenimplementierungen.

Ü 4.3 OMNeT++ Request-Response-Protokoll

Implementieren Sie basierend auf Ü 4.2 ein simples Request-Response-Protokoll vergleichbar mit DNS. Der Client soll in regelmäßigen Abständen einen Request an den Server senden, der nur aus einem Hostnamen besteht. Der Server behandelt den Request und sendet nach einer kurzen Bearbeitungszeit eine Antwort (Response) zurück. Die Antwort besteht aus einem Statuscode (`OK` bzw. `Fehler`) und einer Adresse (4 Bytes). Definieren Sie geeignete Nachrichtenformate fixer Länge für Request und Response.

Halten Sie Ihre Implementierung einfach und verwenden Sie folgende Annahmen:

- Das Intervall zwischen den Anfragen des Clients sei exponentialverteilt mit Erwartungswert `intReq`, wobei `intReq` in der Konfigurationsdatei `omnetpp.ini` konfigurierbar sein sollte.
- Der Client soll insgesamt `n` Anfragen durchführen, wobei `n` ebenfalls via `omnetpp.ini` konfiguriert werden sollte.
- Die Anfrage bestehe aus einer zufälligen Zeichenkette (max. Länge von 50 Zeichen).
- Die Bearbeitungszeit des Servers sei truncated normalverteilt mit μ=`1ms` und σ=`0.2ms`.

Nach der Bearbeitungszeit sendet der Server mit einer Wahrscheinlichkeit von `0.9` eine positive Antwort zurück (Statuscode `OK`). Die Adresse wird dabei zufällig generiert. Ansonsten wird ein negativer Statuscode zurückgesendet und die Adresse erhält den definierten Wert `FFFF FFFF`$_{16}$.

Ü 4.4 aauTCP/IP-Referenzmodell mittels OMNeT++: Kontext und Zielsetzungen

In den folgenden Übungsblättern soll eine vereinfachte Variante des TCP/IP-Referenzmodells, wie in Abbildung 2 dargestellt, mittels OMNeT++ implementiert werden.

TCP/IP-Schicht	~ ISO/OSI-Schicht	Beispiel
Anwendungsschicht	5-7	HTTP, FTP, SMTP
Transportschicht	4	TCP, UDP
Vermittlungsschicht	3	IPv4, IPv6
Netzzugangsschicht	1-2	802.3, 802.11, 802.16

Abbildung 2 — TCP/IP-Referenzmodell

In dieser vereinfachten Variante des TCP/IP-Referenzmodells wird jede Schicht als eigenes OMNeT++-Modul implementiert, wobei nach und nach (d.h. von Übungsblatt zu Übungsblatt) die einzelnen Schichten, beginnend von der Applikationsschicht bis hinunter zur Bitübertragungsschicht[1], implementiert werden. Abbildung 3 gibt einen Gesamtüberblick über das aauTCP/IP-Referenzmodell.

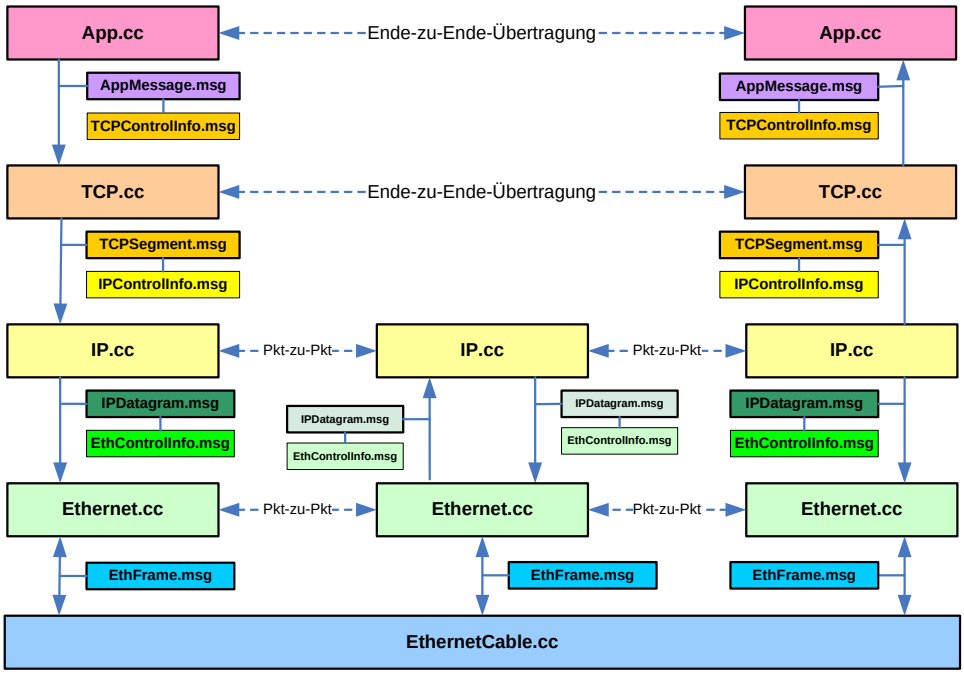

Abbildung 3 — Überblick über das aauTCP/IP-Referenzmodell

Für jede Schicht soll ein eigenes Nachrichtenformat definiert werden, das mittels einer `.msg`-Datei erzeugt wird. In Ihrem Code sollen von Schicht zu Schicht die entsprechenden Nachrichtentypen mit `encapsulate()` eingepackt bzw. mit `decapsulate()` wieder ausgepackt werden.

Als Schnittstelle zwischen den Netzwerkschichten kann jede Nachricht sogenannte Steuerungsinformationen (`ControlInfo`) mitführen, die mit `setControlInfo()` und `removeControlInfo()` angehängt bzw. entfernt werden können.

Achtung: Steuerungsinformationen sind nicht wirklich Teil einer Nachricht und sollen auch nicht über das Netzwerk verschickt werden! Sie dienen nur der Kommunikation zwischen den Schichten auf einem Knoten (Host, Router). Dies spiegelt sich auch in Abbildung wider, wobei die C++-Klassen (`.cc`) die einzelnen Schichten darstellen (zu jeder Schicht ist i.a. auch eine `.ned`-Datei zugeordnet) und die `.msg`-Dateien die Nachrichtenformate und Steuerungsinformationen.

[1] Diese Schicht wird nur sehr vereinfacht realisiert werden.

Jede Schicht wird als einzelnes Modul realisiert. Nachrichten werden zwischen Modulen auf die gleiche Weise ausgetauscht wie zwischen zwei Netzwerkknoten. Das aauTCP/IP-Referenzmodell verwendet `.msg`-Dateien um spezielle Nachrichten (d.h. `AppMessage`, `TCPSegment`, `IPDatagram`, `EthFrame`, etc.) und Steuerungsinformationen zu erstellen (d.h. `TCPControlInfo`, `IPControlInfo`, `EthControlInfo`, etc.).

Die Verzeichnisstruktur ergibt sich aus den einzelnen Schichten und diversen Hilfskonstrukten:

- **3rdParty**: Hilfsklassen aus dem INET-Framework[2], welche u.a. Klassen für IP-Adressen und MAC-Adressen zur Verfügung stellen.
- **app**: Relevante Dateien für die Applikationsschicht wie z.B. Server- und Clientimplementierung sowie das Nachrichtenformat für die Applikationsnachricht.
- **tcp** bzw. **udp**: Relevante Dateien für die Transportschicht wie z.B. die eigentliche TCP/UDP-Implementierung, die dazugehörige Steuerungsinformation sowie das Segmentformat für die Transportschicht.
- **ip**: Relevante Dateien für die Netzwerkschicht wie z.B. die eigentliche IP-Implementierung, die dazugehörige Steuerungsinformation sowie das Datagrammformat für die Netzwerkschicht.
- **eth**: Relevante Dateien für die Sicherungsschicht wie z.B. die eigentliche Implementierung der Sicherungsschicht, die dazugehörige Steuerungsinformation sowie das Frameformat für die Sicherungsschicht.
- **networks**: .ned-Dateien für die Netzwerkdefinitionen.

Ü 4.5 aauTCP/IP-Referenzmodell mittels OMNeT++: HTTP

Machen Sie sich mit HTTP/0.9 vertraut. Implementieren Sie einige einfache Request-Response-Nachrichten zwischen einem HTTP-Client und einem HTTP-Server. Orientieren Sie sich dabei an folgendem Beispiel oder nutzen Sie Ihre Erkenntnisse aus einer vorangegangenen Übung:

```
C: GET /demo/\r\n
S: <html>\n
   \t<head><title>Demo Web Site 2019</title></head>\n
   \t<body>\n
   \t\t<img src="logo.gif"/>\n
   \t\t<h1>Welcome</h1>\n
   \t\t<img src="TechnikErleben.png"/>\n
   \t</body>\n
   </html>\n

C: GET /demo/logo.gif\r\n
S: logo.gif

C: GET /demo/TechnikErleben.png\r\n
S: TechnikErleben.png
```

Implementieren Sie nur jene Teile von HTTP/0.9, die für die Simulation dieses Beispiels notwendig sind! Andere Befehle und Funktionen können vereinfacht implementiert oder ganz weggelassen werden. `HTTPClient` und `HTTPServer` repräsentieren die Applikationsschicht (siehe Abbildung 4).

[2] http://inet.omnetpp.org/

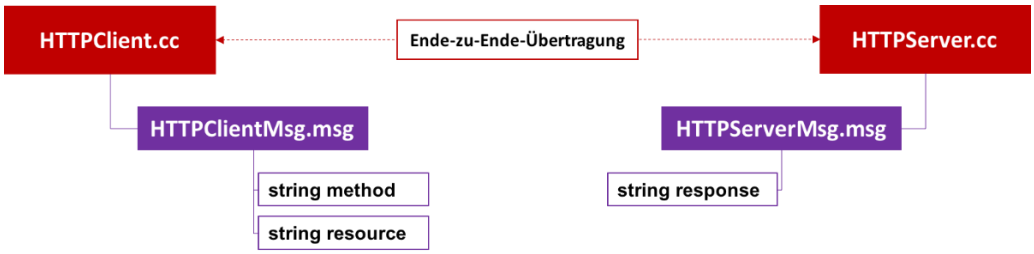

Abbildung 4 — Kommunikation zwischen Client und Server (sinngemäß)

Die Nachrichten, die der Client an den Server sendet (`HTTPClientMsg.msg`), bestehen aus zwei Feldern: (1) `method` und (2) `resource`. Client-seitig wird `method` auf die jeweilige HTTP-Anfrage (z.B. `GET`) und `resource` auf die benötigten Parameter (z.B. `/demo/\r\n`) gesetzt.

Die Antworten des Servers (`HTTPServerMsg.msg`) bestehen nur aus einem Feld: (1) `response`. Dieses wird auf die jeweils angeforderte Ressource gesetzt (Ressourcen werden in diesem Beispiel der Einfachheit halber durch Strings dargestellt).

Eine Reihe von Dateien steht Ihnen zur Verfügung, auf denen Sie Ihre Simulation aufbauen können (app-Verzeichnis):

- `HTTPClient.ned` bzw. `HTTPServer.ned`: Eine einfache Definition eines HTTP-Knotens. Hier müssen Sie nichts ändern.
- `HTTPClientMsg.msg` bzw. `HTTPServerMsg.msg`: Die Definition der Nachrichten, die zwischen Client und Server ausgetauscht werden. Ergänzen Sie diese Dateien um die benötigten Felder (gemäß Abbildung 44).
- `HTTPNetwork.ned`: Die Definition des Netzwerks, das simuliert werden soll. Erstellen Sie in dieser Datei ein Network `HTTPNetwork`, das zwei HTTP-Knoten (vom Typ `HTTPClient` bzw. `HTTPServer`) beinhaltet. Verbinden sie die beiden Knoten durch einen Kanal mit einer Verzögerung von 100ms.
- `HTTPClient.cc` bzw. `HTTPServer.cc`: Die Implementierung der Funktionalität des Clients bzw. des Servers. Ergänzen Sie die Funktionalität für das Empfangen und Senden der Nachrichten. Geben Sie den Inhalt der empfangenen Nachricht auf der Konsole aus.

Hinweise:

- Vergessen Sie nicht auf die omnetpp.ini-Datei, sonst lässt sich Ihre Simulation nicht starten.
- Implementieren Sie die gewünschte Funktionalität ausschließlich in den Dateien `HTTPClient.cc`, `HTTPClient.h`, `HTTPClientMsg.msg`, `HTTPServer.cc`, `HTTPServer.h`, `HTTPServerMsg.msg` und `HTTPNetwork.ned` in den, mit entsprechenden Kommentaren gekennzeichneten, Quellcodebereichen. Zusammen mit den anderen Dateien ergibt das die vollständige Simulation.
- Sollten die Sourcedateien für `<name>.msg` Dateien nicht automatisch erzeugt werden, so generieren Sie diese manuell auf der Konsole (mingwenv.cmd im OMNeT++ Installationsverzeichnis) mit zum Beispiel folgendem Befehl:

```
opp_msgc <pfad zum projekt>/src/app/<name>.msg
```