Research Report 5/00

Modeling and Analysis of Wireless Network Segments with aid of Teletraffic Fluid Flow Models

by
Markus Fiedler

Department of Telecommunications
and Signal Processing
University of Karlskrona/Ronneby
S-372 25 Ronneby
Sweden

ISSN 1103-1581
ISRN HKR-RES—00/5—SE

Modeling and Analysis of Wireless Network Segments with aid of Teletraffic Fluid Flow Models

by Markus Fiedler

ISSN 1103-1581
ISRN HKR-RES—00/5—SE

Printed by Psilander Grafiska, Karlskrona 2000

Research Report

Modeling and Analysis of Wireless Network Segments with aid of Teletraffic Fluid Flow Models

Markus Fiedler
University of Karlskrona/Ronneby
Dept. of Telecommunications and Signal Processing (ITS)
S-371 79 Karlskrona
markus.fiedler@its.hk-r.se

990615 – 993011

Sponsored by:
T-Nova Deutsche Telekom AG
Technologiezentrum
Am Kavalleriesand 3
D-64295 Darmstadt

German title:
Modellierung und Analyse drahtloser Netzsegmente
mit Hilfe verkehrstheoretischer Fluß-Modelle

Abstract

The fluid flow model is used to model variable server and link rates, as they appear in mobile channels due to fading, bit error recovery and failed channel reservations. Assuming a Gilbert-Elliott model for the channel, the influence of transmission quality on network Quality of Service (QoS) might be studied. Thus, the fluid flow model assumes the role of a model that might be used for dimensioning and performance evaluation both at the edge and inside a network. We present a user-friendly, contemporary, flexible, fast and numerically stabilized computing environment for the fluid flow model with a well-known user interface that is able to handle multiple users and that might be used as well for batch processing. We discuss two case studies that emphasize the crucial impact of the relationship between server and source dynamics on QoS.

Keywords: Fluid flow model, variable server capacity, Gilbert-Elliott model, performance analysis, QoS, software engineering, numerical methods

Acknowledgement

The help of Dipl.-Ing. Holger Voos, Institute of Process Automation, Universität Kaiserslautern, Germany, is gratefully acknowledged. This work gained a lot from the fruitful discussions during his visit in Karlskrona in August 1999.

Contents

Chapter 1

Introduction

The fluid flow model is a model that captures all kinds of effects that originate from time-variable bit rates in all kinds of networks. Its importance in communications came up at the same time as the *Asynchronous Transfer Mode* (ATM) was invented. ATM offers an outstanding possibility to allocate network resources in a very flexible way and combining advantages of circuit- and packet-switched networks. Especially for connection-oriented traffic that exhibits *variable bit rate* (VBR), ATM is able to cope with so-called *over-booking*: a link carrying a certain number of connections does not necessarily need to provide the total peak cell rate of all connections, because each single connection does not need to use its full share of the link all of the time. Indeed, such a *multiplexing gain* is a very natural thing for packet-switched networks, especially when the packets share the link in a statistical manner. Under ideal conditions, the capacity allocation might be reduced from peak rate allocation to (a little bit more than) mean rate allocation. However, this multiplexing gain is not achievable for *constant bit rate* (CBR) traffic, for which peak and mean bit rate are the same. Of course, the fluid flow model is able to deal with this quite simple case.

The main aspect which the fluid flow model is able to model is addressed by the following question: What happens if the arrival bit rate to a traffic *concentrator* (or *multiplexer*) or to a simple *buffer* temporarily exceeds the service rate? Such situations might be due to:

1. A varying input rate that might temporarily become higher than the output rate due to VBR input traffic streams

2. A varying output rate that might temporarily become lower than the peak requirement of input traffic streams

Situation 1 is typical for the exploitation of multiplexing gain both at the edge and inside the core network. The fluid flow model might help to dimension network links in a way to allocate as little capacity as possible while maintaining the desired *Quality of Service* (QoS), thus maximizing the gain from operating that link. In connection to wireless network segments, we are especially interested in situation 2, which may arise in the following scenarios:

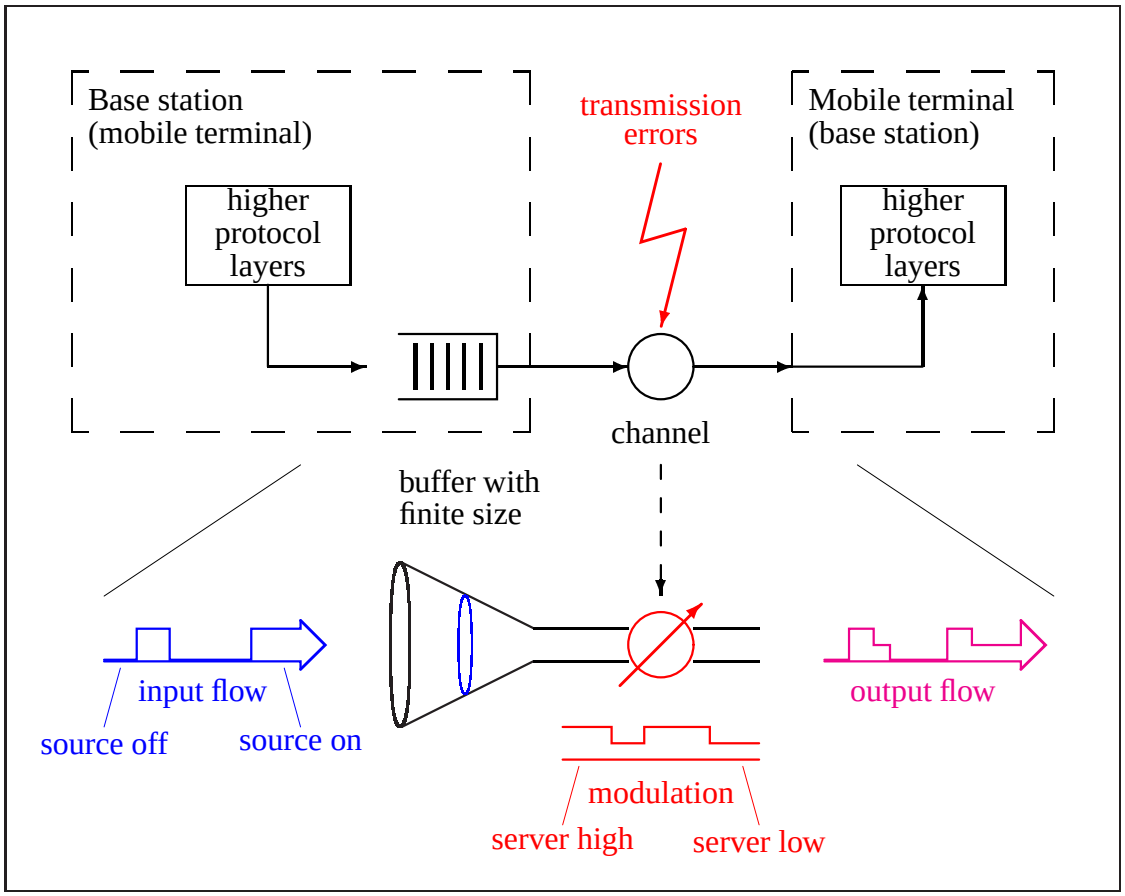

Figure 1.1: Fluid flow model of a wireless link: Influence of transmission quality on server capacity.

a. A channel has states of different quality (Gilbert-Elliott model), which causes the effective channel bit rate to vary over time. If the channel is “bad”, queuing delays and perhaps even data loss become worse [Tur-93, Kim-98, Fie-00b]. Figure 1.1 shows the corresponding scenario.

b. A base station or a mobile terminal does not succeed in reserving time slots for a GPRS packet to be sent. This means that the channel is temporarily unavailable, and the packet has to be queued. Also in this case, queuing delays and the risk for data loss grow. A question to be answered in this context would be how to dimension a buffer in order to reduce the loss ratio below a desired level [Kim-98, Fie-00b].

c. The capacity that is available for a certain traffic stream varies because there is priority traffic “sharing” the same link that is able to use as much capacity as it needs temporarily. A typical situation at the edge of the network is the competition between voice and data traffic for time slots in GSM/GPRS cells, where data traffic has to take the “left-over” from voice traffic. But also inside the core network, similar problems might arise. With

the introduction of QoS concepts to Internet, such questions will surely gain more and more significance within the next years.

d. In AAL-2, there exists the possibility of adapting the coding to network load conditions: When the content of a network buffer reaches certain levels, one or two of the least significant bits are thrown away. In principle, this is a virtual increase of capacity which is controlled by the buffer content to keep the queue from growing. This measure is typical for carrying mobile conversations over an ATM core network [Lam-97].

e. *Traffic shaping*, e.g. according to the *Leaky Bucket* algorithm, aims at improving the utilization of the network by making traffic patterns more useful for statistical multiplexing. The shaper itself is modeled as a buffer with variable capacity, which is controlled either by feedback from the network (ABR) or by local feedback from a token pool. Shaping usually happens at the edge of the network [Sch-98].

f. Network reliability problems, e.g. *link breakdown*, is an issue for network dimensioning and control both at the edge and inside the network [Kro-99].

All this shows the outstanding role of the fluid flow model for end-to-end networking, because it provides a unifying framework for network performance evaluation.

On this background, the research project that is described in this report aims at providing a user-friendly and numerically stable tool for carrying fluid flow analysis. Section 2 glances at fluid flow history and describes the fluid flow modeling of sources, buffer and server. Some general aspects of the analysis and its aims are discussed, and the notation that is used throughout the project is defined. Section 3 explains basic and innovative concepts that have been used for implementing the fluid flow analysis, e.g. the web/compute server concept. Section 4 describes both the main steps in fluid flow analysis and the software objects, attributes and methods/functions to carry them out. Section 5 presents two case studies, one which deals with scenarios a and b [Fie-00b], and another one that shows quite interesting results for scenario f [Fie-00a]. Section 6 summarizes the project and gives an outlook on future work; it is followed by the bibliography and the appendix that contains some details on the software that has been developed.

Chapter 2

The Fluid Flow Model

2.1 History

This subsection mentions some important milestones in fluid flow analysis that are important in the context of this project.

- 1960's: The fluid flow model appears within manufacturing [Mit-88].
- 1974–1988: Pioneering work happens within communications [Kos-74, Ani-82, Kos-84, Tuc-88, Ste-91].
- 1988: Mitra studies a manufacturing system with more than one server, i.e. variable server capacity [Mit-88].
- 1990–1995: Many approximations (e.g. based on infinite buffer sizes) and "equivalent bandwidths" (used for capacity dimensioning) appear [Gué-91, Elw-93, Kon-94] *et al*.
- 1995: Yang *et al* admit numerical problems for quite small systems [Yan-95].
- 1997: Fiedler and Voos identify sources for the numerical problems [Fie-97, Fie-99].
- 1998: Kim and Krunz use the fluid flow model in a mobile environment [Kim-98].
- 1999: Kroese and Nicola use the fluid flow model for a simulation study of breakdowns in queues [Kro-99]. Ramaswami presents matrix-geometric methods for fluid queues with infinite buffer [Ram-99].

Even today, the numerical side of the fluid flow model has a bad reputation. However, the work [Fie-97, Fie-99] on which this project is based shows that it is possible to overcome the corresponding difficulties.

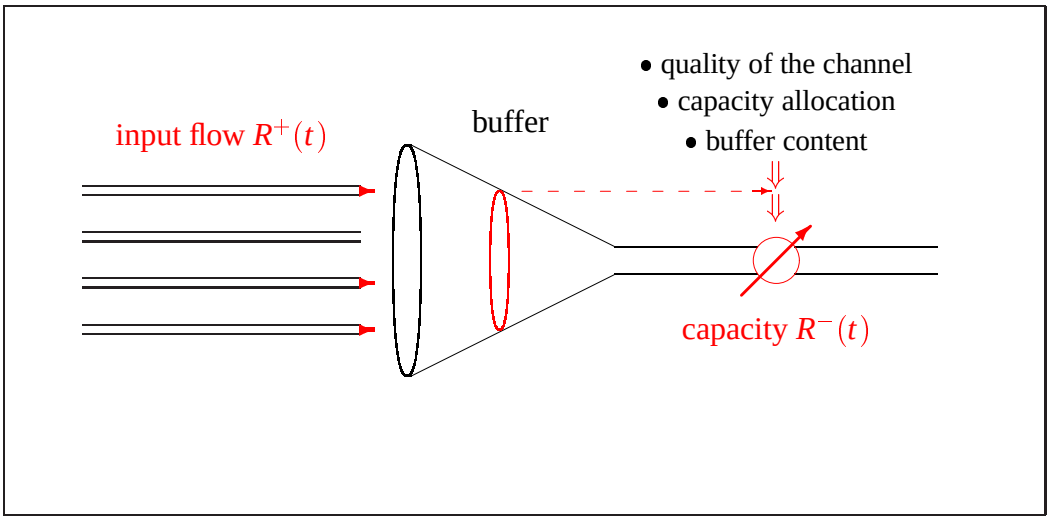

Figure 2.1: Fluid flow model of a multiplexer.

2.2 Multiplexer Model

The fluid flow model of a multiplexer is shown in Figure 2.1. It takes a couple of influences on its capacity into account.

All information streams are modeled by streams of fluid from sources to sinks. At the entrance of the buffer, fluids are superimposed if more than one source exists or is active. In the general case, both *total input rate* $R^+(t)$ and *total service rate* $R^-(t)$ are random processes, whose properties are basically described in sections 2.3 to 2.7. The index $+$ stands for the fact that a source "adds" fluid to the buffer, while a server "takes away" fluid from the buffer (index $-$).

Before we begin to look at the description of the source characteristics and the source parameters, we have to discuss the role of data and time units. One of the advantages of the fluid flow model consists in the fact that data and time units might be freely chosen. Of course, the same data and time units have to be used throughout the whole model: The units of input and server rates have to be matched, and there are also connections to transition rates and the buffer size K. Thus, the user has the freedom to choose units, but has the responsibility to stick to them! This issue is discussed in the manual pages (see appendix).

2.3 Source modeling

2.3.1 Groups of sources

Sources that have the same parameters may be grouped. The affiliation of a source to a certain group is expressed by the index i. The advantage of grouping will become obvious later when it comes to the size of the overall state space ν. The number of groups is denoted by g.

We shall look at the superposition of several groups of sources and servers in section 2.6.

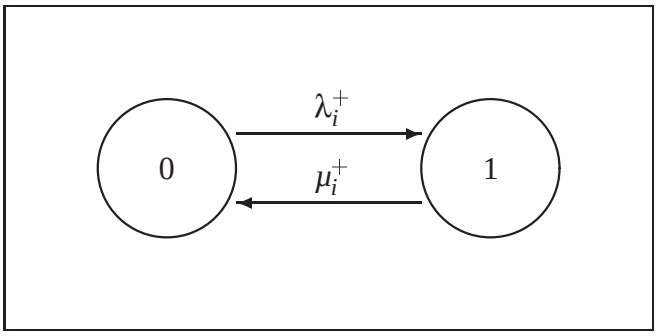

Figure 2.2: State diagram for one VBR 2-state source. State 0 refers to the "source low" state with bit rate l_i^+, while state 1 refers to the "source high" state with bit rate h_i^+.

2.3.2 VBR 2-state sources

A *VBR 2-state source* has one "low" and one "high" state, both with exponential duration. The state diagram shown in Figure 2.2 captures the way a source of this kind contributes to the total input rate $R^+(t)$.

The source parameters are as follows:

1. *Input rate in "source low" state* l_i^+, given in data units per time unit.
2. *Input rate in "source high" state* h_i^+, also given in data units per time unit.
3. *Transition rate from "source low" state to "source high" state* λ_i^+, given in reciprocal time units.
4. *Transition rate from "source high" state to "source low" state* μ_i^+, also given in reciprocal time units.

A special case is the *on-/off source*. Such a source is inactive ("off") during its "low" phase, i.e. $l_i^+ = 0$, and active ("on") during its "high" phase.

From the parameters, we obtain a couple of other parameters. The *mean duration of a "low"/"high" phase* is given by $(\lambda_i^+)^{-1}$ and $(\mu_i^+)^{-1}$, respectively. The *mean cycle time for a source* is given as

$$\tau_i^+ = \frac{1}{\lambda_i^+} + \frac{1}{\mu_i^+}, \tag{2.1}$$

which means that on average, one "low" and one "high" phase pass during time τ_i^+. Furthermore, we define the *source (high-) activity factor* (the term in brackets is omitted for on-/off sources)

$$\alpha_i^+ = \frac{\lambda_i^+}{\lambda_i^+ + \mu_i^+} \tag{2.2}$$

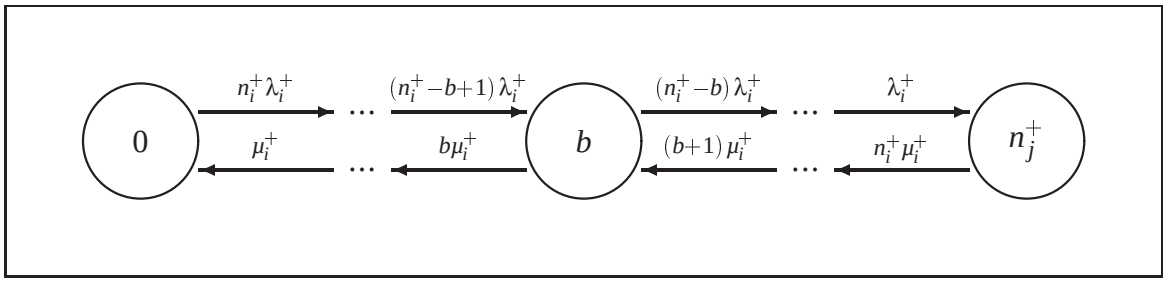

Figure 2.3: State diagram for a group of n_i^+ VBR 2-state sources. The number of the state b indicates how many sources are in the "high" state.

which represents the *state probability for the "high" ("on") state*. The smaller α_i^+ becomes, the shorter the "high" phases get on average compared to the mean cycle time.

If τ_i^+ and α_i^+ are preferred to characterize the behavior of a source, the following equations might be applied:

$$\lambda_i^+ = \frac{1}{(1-\alpha_i^+)\,\tau_i^+} \tag{2.3}$$

$$\mu_i^+ = \frac{1}{\alpha_i^+\,\tau_i^+} \tag{2.4}$$

The superposition of n_i^+ VBR 2-state independent sources within a group leads to a state diagram representing a Markov chain with

$$\nu_i^+ = n_i^+ + 1 \tag{2.5}$$

states as it is shown in Figure 2.3. The state probabilities $\vec{\pi}_i^+$ are binomially distributed, i.e.

$$\pi_{s,i}^+ = \binom{n_i^+}{s_i^+} (\alpha_i^+)^{s_i^+} (1-\alpha_i^+)^{n_i^+ - s_i^+} . \tag{2.6}$$

The input rate of the whole group of sources in state s is given by

$$r_{s,i}^+ = (n_i^+ - s_i^+)\, l_i^+ + s_i^+\, h_i^+ \tag{2.7}$$

and the *mean input rate* by

$$m_i^+ = n_i^+ \frac{\mu_i^+ l_i^+ + \lambda_i^+ h_i^+}{\mu_i^+ + \lambda_i^+} , \tag{2.8}$$

which in the on-/off case becomes

$$m_i^+ = n_i^+\, \alpha_i^+\, h_i^+ . \tag{2.9}$$

2.4 CBR sources

CBR sources are somehow a special case of VBR 2-state sources: either the "low" state is never assumed ($\mu_i^+ \to \infty$), or both states have the same input rate ($l_i^+ = h_i^+$). Thus, the contribution of a CBR source to the total input rate $R^+(t)$ is completely described by the constant

1. *input rate (in "source high" state)* h_i^+, given in data units per time unit.

Independently of the number of sources in the group, the state space has the size of one:

$$\nu_i^+ = 1 \tag{2.10}$$

This state occurs with probability 1.

2.5 Server modeling

The modeling of the servers that contribute to the total server rate $R^-(t)$ is basically the same as for the corresponding sources. To avoid repetitions, we use a fully symmetrical notation, i.e. we apply the same definitions as before and exchange the index $^+$ by the index $^-$.

Moreover, we restrict ourselves to one group of servers of the same kind, i.e. we may omit the index *i*.

2.5.1 VBR 2-state server

The server parameters are as follows:

1. *Server rate in "server low" state* l_i^-, given in data units per time unit.
2. *Server rate in "server high" state* h_i^-, also given in data units per time unit.
3. *Transition rate from "server low" state to "server high" state* λ_i^-, given in reciprocal time units.
4. *Transition rate from "server high" state to "server low" state* μ_i^-, also given in reciprocal time units.

Instead of λ_i^- and μ_i^-, the *mean cycle time for a source* τ_i^- and the *state probability for the "high" state* α_i^- may be used, for example. Especially the mean cycle time gives an impression on how fast a server changes states compared to a source.

The connection with the Gilbert-Elliott model may been seen as follows: In the Gilbert-Elliott model, a channel has — at least — two states, one "bad" and one "good" state, see [Tur-93, Kim-98] and Figure 2.1. While the channel is fully available when it is "good", it looses a part or even the whole of its capacity in the "bad" state, which is mainly due to bit errors as a result of different influences, e.g. a temporarily bad signal-to-noise ratio.

2.5.2 CBR server

A CBR server has been the standard for fluid flow analysis for a long time. It is fully described by the

1. *server rate in "server high" state* h_i^-, given in data units per time unit.

2.6 Superposition of sources and servers

All the source states for group i are collected in the column vector $\vec{s}_i^+$. Out of these vector, the source state vector for all groups is composed by using Kronecker algebra:

$$\vec{s}^+ = \vec{s}_1^+ \otimes \ldots \otimes \vec{s}_g^+ \,. \tag{2.11}$$

This is a kind of formal composition, where the "+" symbol between the group-related states points out that the corresponding states occur together. To capture all states of the system, the server state vector $\vec{s}^-$ is joined in the same way. The result is the state vector for the whole system:

$$\vec{s} = \vec{s}^+ \otimes \vec{s}^- \tag{2.12}$$

For the size of the overall state space, we obtain the expression

$$\nu = \nu^+ \nu^- = \nu^- \prod_{i=1}^{g} \nu_i^+ \,. \tag{2.13}$$

Assuming independence, the *state probabilities* are given in closed form. For the vector element s, in which the groups of sources are in the states $s_i^1, \ldots, s_i^g$ and the servers are in the state s^-, we obtain

$$\pi_s = \pi_s^- \prod_{i=1}^{g} \pi_{s,i}^+ \,. \tag{2.14}$$

2.7 Buffer modeling

The buffer may be modeled as a funnel or a bucket with a hole representing the outgoing link that is fed by one or more sources. Its content, $X(t)$, depends on the balance between the total input rate $R^+(t)$ and total server rate $R^-(t)$, which is called *drift*

$$D(t) = R^+(t) - R^-(t) \,. \tag{2.15}$$

The drift depends on the state in which sources and servers are. Thus, it takes on the values

$$d_s = r_s^+ - r_s^- \quad \forall s \,. \tag{2.16}$$

Depending on the drift, we may observe one of the behaviors described in the next subsections.

2.7.1 Positive drift and over-load states

If the drift is positive ($d_s > 0$) , i.e. the total input rate exceeds the total service rate in state s, the buffer content rises by $\Delta X = d_s \Delta T$ during the time ΔT as long as the buffer does not get full not full. Therefore, all states that have positive drift constitute the *set of over-load states* S^o.

Once the buffer gets full, the drift describes the total loss rate, i.e. during ΔT, the amount of fluid ΔX gets lost. Loss and its distribution is discussed more detailed in subsection 2.7.4.

2.7.2 Zero drift and equilibrium states

Zero drift ($d_s = 0$), i.e. matched total input and service rate, freezes the level of buffer content as long as this state is kept. The corresponding states constitute the *set of equilibrium states* S^e. No loss happens.

2.7.3 Negative drift and under-load states

If the drift is negative ($d_s < 0$), i.e. the total server rate exceeds the total input rate in state s, the buffer content either diminishes by ΔX during ΔT, or it remains zero. Such states are to be found within the *set of under-load states* S^u and do not lead to any loss at all.

2.7.4 Loss and its distribution

We have seen that loss merely occurs under the following conditions:

- The system is in an overload state.
- The buffer is full.

Loss is distributed in a fair way among the groups, which is based on their momentary input rate. The loss rate becomes

$$r_{i,s}^{\mathrm{L}} = \frac{r_{i,s}^{+}}{r_s^{+}} d_s \,. \tag{2.17}$$

The application of this principle to one group shows that loss is distributed evenly between the information streams belonging to that group. However, information streams of different groups may experience different loss probabilities.

2.7.5 The roles of sources and servers

From definitions (2.15f), it becomes clear that from the drift point of view, it does not matter whether the total input rate rises (or sinks) by Δr of the total service rate sinks (or rises) by Δr. *The servers act as sources with negative data rates.* This has been one of the reasons for choosing our symmetrical notation. However, this symmetry does not hold for everything — servers do not experience any loss.

2.8 Model variants and load conditions

2.8.1 Load definition

The *offered load* is defined as the ratio of the total mean input rate and the total mean service rate, to which the different groups of sources contribute in an additive way:

$$A = \frac{m^+}{m^-} = \sum_{i=1}^{g} \frac{m_i^+}{m^-} = \sum_{i=1}^{g} A_i \tag{2.18}$$

2.8.2 Model with infinite buffer

The buffer is not of limited size ($K \to \infty$). Thus, there is no loss at all, but on the other hand, queuing delays might get extremely high. This case has been studied extensively in literature, because it allows for certain simplifications of the analysis. However, infinite buffer sizes are impossible in practice.

The goal of the analysis consists in finding the *complementary probability distribution function of the buffer content* (cpdf) $G(x) = \Pr\{X > x\}$. A buffer content of x is henceforth called *buffer threshold*. For one group of VBR 2-state sources and a CBR server, there exists a very elegant method [Ani-82], which unfortunately cannot be used for the general case of different groups in combination with VBR-2 state servers.

The offered load has to be limited to $A < 1$ (or $A \leq 1$, if all sources and servers are of the CBR type).

2.8.3 Model with finite buffer

This is the most general case ($0 \leq K < \infty$), which at the same time is hardest to analyze. It is hardly dealt with in literature due to effort and numerical problems. For a group i of *on-/off sources*, the buffer size may be given in κ_i mean burst sizes:

$$K = \kappa_i \frac{h_i^+}{\mu_i^+}\,. \tag{2.19}$$

As the analysis of this model mostly focuses on loss-related parameters, the most interesting outcomes are the *probabilities that the buffer is full in overload states* u_s^+. For buffer thresholds $x < K$, it might be interesting to look at the cpdf $G(x)$ to get an idea on how delays are distributed.

Essentially, the offered load is not limited, but the mainstream in the analysis relies on $A < 1$, which simplifies the handling of the states very much.

2.8.4 Buffer-less model

This model is obtained by passing the buffer size to the limit zero ($K \to 0$), which implies that the buffer looses its buffering capability, so that loss happens almost directly after the system has entered an overload state. Such a behavior is typical for systems with very low transition rates, i.e. systems that change states very slowly compared to the dynamics of the buffer.

In this limit, the probabilities that the buffer is full in overload states are identical to the *state probabilities for the overload states* themselves. The fact that state probabilities are given in closed form, c.f (2.6) and (2.14), makes the analysis fast and numerically stable. On the other hand, the cpdf $G(x)$ cannot be considered anymore.

Also in this system, the load is not limited, neither in theory nor in the implementation.

2.9 Quality of Service-related parameters

Before looking at the Quality of Service (QoS)-related parameters, we have to point out a very basic condition that prevents the fluid flow model from delivering zero values: *The minimal total server rate has to be smaller than the maximal total source rate.* If this is not the case, the buffer is never able to fill, and no loss will ever happen. The fact that even systems of CBR sources and servers with matched rates (i.e. 100 % load) do not fulfill this criterion underline the importance of the VBR property in the context of the fluid flow model.

2.9.1 Probability of saturation

The *probability of saturation* describes the probability that the total input rate exceeds the total server rate:

$$P_{\mathrm{Sat}} = \Pr\{R^+ > R^-\} = \sum_{s \in \mathcal{S}^o} \pi_s \tag{2.20}$$

This probability is a measure for the share of (over-load) states that may lead to a degradation of QoS, no matter how the buffer is able to handle this. So from the model's point of view, the buffered models need not be considered. Furthermore, this parameter does not reflect the viewpoint of different groups.

2.9.2 Loss probability

The *individual loss probability* for group i describes the probability that a fluid particle that is belonging to this group gets lost:

$$P_{\mathrm{Loss},i} = \frac{1}{m_i^+} \sum_{s \in \mathcal{S}^o} u_s^+ \, r_{i,s}^{\mathrm{L}}, \tag{2.21}$$

where the loss rate is given in (2.17). For the *total loss probability*, we obtain

$$P_{\text{Loss}} = \frac{1}{m^+} \sum_{s \in \mathcal{S}^o} u_s^+ \, d_s \,, \tag{2.22}$$

which is a weighted mean of the individual loss probabilities:

$$m^+ \, P_{\text{Loss}} = \sum_{i=1}^{g} m_i^+ \, P_{\text{Loss},i} \,. \tag{2.23}$$

In the buffer-less case, the probabilities that the buffer is full in (over-load) states $s \in \mathcal{S}^o$, u_s^+, merely needs to be replaced by the corresponding state probabilities π_s^+, i.e. the loss probabilities are described by closed formulae.

2.9.3 Overflow probability of a buffer threshold

The *overflow probability of a buffer threshold x* is described by the value of the complementary buffer content distribution $G(x)$. Observe that $G(x \geq K) = 0$, which means that only the buffered models are of interest. The parameter does not take the different views by different groups of sources into account.

Chapter 3

The Fluid Flow Calculator

This chapter describes the basic software concepts, while implementation details like class attributes and methods are mentioned in chapter 4. Details about the use of the software may also be found in the user's manual web pages, which are to be found in the appendix.

Most self-written tools suffer from the lack of a *Graphical User Interface* (GUI) that makes it possible to use the tool in an intuitive way. Mostly, such tools offer very little flexibility with regard to data in- and output, and sometimes, the user is forced to learn (a vast amount of) cryptic commands or even to modify source code to get the desired functionality. So first, we shall look at some important aspects of the GUI and how this affects the overall software concepts. Later, we will focus on the very heart: the C++ calculation program.

3.1 Graphical User Interface (GUI)

3.1.1 Requirements

1. The GUI should make it easy to use the calculation program.
2. The GUI should use elements and a functionality that the user is familiar with.
3. The GUI should be platform-independent while keeping its functionality.
4. The GUI should use current technology, which makes it easier to maintain the interface and to adapt it to forthcoming developments.
5. The GUI should not affect the performance of the calculation program.
6. The GUI should make it easy to integrate manual and help functions.

3.1.2 Alternatives

1. *Microsoft (MS) Windows-specific solution (Visual Basic, e.g.).* As the MS Windows-based PC has the leading position among the end-user computers, most people are acquainted with MS GUIs. However, the GUI-specific part of the code is mostly not portable, even between different versions of the MS Windows OS. A re-compilation of the DLL files is necessary, which often comes along with the need to adapt or rewrite parts of the code. As a new Windows release appears on average each second to third year, this could imply a lot of adaptation work in the future. Another disadvantage is the need for embedding the GUI functions into the code, which could affect the speed of a calculation in a negative way.

2. *X-Windows-based solution.* X is a quite widespread standard for GUIs, but mostly within the UNIX/LINUX environment. Therefore, this solution does not seem to be feasible for Windows-based PCs and thus neither for the whole project.

3. *HyperText Markup Language (HTML)-based solution.* Here, a browser window is used as GUI. As HTML is designed to work independently of the OS, a browser-based solution should work quite independently of the underlying OS (Windows, LINUX, UNIX and others), as long as the current standards are supported. Moreover, this solution allows for a full separation of GUI and calculation program in the sense of a client-server solution, which will be discussed in the next section.

3.2 The web/compute server concept

The *web/compute server concept* represents a client-server solution that is

- flexible,
- simple to handle,
- almost independent of the underlying operating systems,
- capable of separating calculation and GUI tasks,
- multi-user friendly.

3.2.1 The server

The notion of the concept already shows that the role of the server is two-fold:

1. The main task for the server is to be a *compute server*, i.e. to execute the calculation program in a fast and reliable way. Thus, a powerful computer is needed, e.g. with UNIX or LINUX operating system.

2. The other task consists in providing communication facilities between the calculation program and its users. To achieve this, the compute server has to become an (Intra- or Internet) *web server*.

Observe that the task of being a web server is restricted in

1. executing scripts,
2. sending their HTML output back to the clients,
3. starting programs and
4. making the files produced by those programs accessible to the user.

To be able to do that, the following web server configurations have to be set:

- The server should be enabled to execute *Common Gateway Interface (CGI) scripts*.
- The timeout variable should be set in a way that the web server is patient enough to wait for calculations to complete.
- User `nobody` that actually represents a remote user accessing the web server should be allowed to carry out all the scripts and programs as well as to access and write files in the corresponding directory.

Besides the code attached to this project, see section 3.3, the server requires access to or installation of the following software:

- a `Perl` interpreter;
- a C++ runtime library for the execution of the calculation program;
- the public-domain plot program `gnuplot`, if PostScript output is desired.

More on installation and application is to be found in the appendix.

On the other hand, the concept of a web/compute server implies that this server should be able to carry out more than one calculation process at the time. For contemporary computers and operating systems, this should not be any problem. But the more processes are carried out at the same time, the longer each calculation takes. Therefore, it is recommended to limit the user group that might connect to a certain web/compute server.

3.2.2 The clients

The role of each client, i.e. browser, consists in representing the GUI to the user, while the server is almost relieved from that — besides from producing the HTML code to be displayed by the browser. The concept allows more than one client connecting itself to the server at the same time. The clients themselves (Microsoft Explorer or Netscape Communicator, e.g.) work almost independently of the underlying operating system or hardware. The typical end-user system consists in a Windows PC.

3.2.3 HTML-based solutions

An HTML-based solution might use different technologies.

1. *Java client-server solution*: A local Java applet or a Java script that is embedded in HTML page allows for interaction with the user and communicates with a Java application on the server that interacts with the C++ calculation program. The calculation program should not be written in Java, because this would lead to a dramatic slow-down of the calculations. Moreover, some browsers still have problems with executing Java.

2. *Java client-C++ socket solution*: Here, the Java client communicates directly with the C++ calculation program via a socket, which is the only difference to scenario 1.

3. *CGI (Common Gateway Interface) solution*: A local FORM that is embedded in a HTML document receives data from the user and activates a script (batch, shell or Perl, e.g.) on the server that calls the calculation program and sends HTML-related output back to the client, e.g. links to files that have been produced.

The advantage of scenario 2 consists in enabling the user to communicate with the calculation program in an interactive way. However, for standard applications, such an interaction is not needed; the next section contains a discussion of this topic. Due to its universality and simplicity, scenario 3 has been chosen. The HTML code and the CGI script are easy to understand and modify. Furthermore, the CGI script is compiled by Perl on-the-fly, which means that the only program to be compiled manually in case of a change is the calculation program itself. The parts of the software are described in the following section.

3.3 Software components

3.3.1 HTML form page `ffcalc.htm`

The main tasks of the HTML form page consist in providing the GUI for collecting user input and in sending this input to the CGI script. The latter is done via the POST facility, i.e. the script is called with a couple of command-line parameters that are determined by the variables

used in the HTML form. Observe that different browsers might send the information in different orders; the script `ffcout.cgi` will have to take care of this. The following variable types have been used:

- select — choose from list of items;
- input — ASCII text in general;
- radio — choose from a couple of possibilities;
- submit and reset button.

All the fields are initialized with default values. The user may fill in the fields, which mostly require numerical input, or choose options being offered in whatever order (s)he wants. The structure of the page is as follows:

1. The Quality of Service, the fluid flow model and the buffer-related thresholds have to be chosen.
2. The groups of sources and their parameters have to be defined.
3. The servers have to be specified.
4. The desired kind of output has to be chosen.

The Reset button resets all the fields to their default values. If the user clicks on the Start button, the variable names and their values are passed to the CGI script via its command line.

Further information on the usage of this form and especially on the meaning of the different parameters might be obtained from the manual pages (see appendix), which may be reached directly via a link from the HTML page containing the form.

3.3.2 CGI script `ffcout.cgi`

The CGI script that is called from the HTML form gets its input parameters via command-line arguments. Then,

- it checks the input and issues error messages or warnings;
- it starts the calculation program `ff_cmdln`, if the input data did not contain any obvious errors;
- on success, it issues a link to an Excel-compatible data file that the user may download and displays that file;

- it produces a gnuplot control file and starts gnuplot to write a PostScript file, if the user has chosen that option and if there is enough data to produce a picture;
- it issues links to the desired PostScript picture and the corresponding gnuplot control file.

The error messages and warnings reported by the script are also mentioned in the user manual. There is no link back to the form page to prevent its entries from being deleted when the user should want to make some corrections! To go back, the back button of the browser has to be used.

3.3.3 The original interactive calculation program `ffc`

The original calculation program `ffc` that has been developed between 1995 and 1997 has been designed as a very universal tool, based on the following requirements:

- The program should be able to carry out a whole set of complicated and time-intensive calculations such as the iterative determination of required capacity of a CBR link for a changing population of sources with possibly different QoS requirements.
- The program should allow to step into intermediate calculation results, e.g. to make it possible to search for numerical problems.
- The program should enable the user to carry out different function in a very flexible way.
- The program should make it easy to include new functions.
- The program should be able to operate in batch mode with input/output redirection from/to files.

To be able to cope with these partly contradicting requirements, the functions of the program (about one hundred) are called via commands in a way that is known from most line-oriented tools, e.g. `ftp` or `gnuplot`. Thus, the program has a *command-oriented user interface*. Before carrying out the corresponding functionality, each function checks whether all the required objects exist, thus making the program robust w.r.t. user faults. For developing purposes, this concept has proven to be very useful.

However, for data production purposes, such a command-oriented interface is a kind of second-best solution. Experience has shown that the need for different input files for each new calculation series and the need for editing all these files imply a lot of effort as well as possibilities to introduce erroneous values. As the program has been designed primarily for human interaction, the call from a batch file with erroneous input files could result in program instability and large amounts of useless data on the harddrive.

The files from which `ffc` is to be built are listed in the appendix.

3.3.4 The command-line calculation program `ff_cmdln`

Both the specific disadvantages of the command-line user interface for batch jobs and the observation that certain popular "paths" exist, along which such a program is used, led to a new concept used in the program `ff_cmdln`: The input data is not read by the program via input files containing commands and values for the user-interface, but via the program call itself as command-line parameters. This concept has several advantages:

- Due to the limited number of "paths" through the program, i.e. the fixed semantics, it becomes feasible to check for reasonable input values.
- Input errors can be fixed in the batch file without having to open and edit one more input file.
- Other programs may call `ff_cmdln` without having to write a file containing the necessary commands first.

It has to be stretched that the *same* calculation program might be used both with the GUI and with batch programs. Furthermore, `ff_cmdln` and `ffc` are built from the same sources; the command-line version uses a subset of `ffc`'s functions and source files. This concept enables an advanced user to carry out much more detailed analysis by using the large set of commands that `ffc` offers based on the same source files, i.e. under fully identical circumstances. The corresponding list of source files is to be found in the appendix.

The parameters needed for calling `ff_cmdln` are to be found in the HTML user manual. The calculation program will check each parameter and exit on error with a detailed error message error or at least issue a warning if some parameter value had to be adjusted. This kind of error handling dominates the source code, but makes the program safe for batch processing. The corresponding error codes are also documented in the HTML user manual.

3.4 Output format

The output format has been chosen to be as general as possible. The data file that may be imported into *MS Excel* may as well be used for producing PostScript pictures with help of the public-domain software `gnuplot`. The structure of this file and the steps that are necessary to import the file into Excel are also described in in the HTML user manual.

The ending `.xls` leads to an automatic call of MS Excel on Windows PCs when the user clicks on the link to download the file. The same applies for the PostScript file name ending `.ps`, if a PostScript viewer (e.g. GhostView) has been installed on the corresponding PC. In the latter case, a graph showing the QoS on the y axis with logarithmic scale and the thresholds on the x axis might be displayed in the browser window without any further user interaction, as it is required in Excel. Furthermore, the PostScript file may be used for slides or documents (see Figures 5.3 and 5.4).

3.5 Long double versions

The programs `ffc` and `ff_cmdln` get into numerical trouble if the number of VBR 2-state servers or sources within one group becomes about 200 (or even less for $\alpha \cdot \to 0^+$ or $\alpha \cdot \to 1^-$). This problem, which is the most critical numerical issue for fluid flow analysis, is also referred to in section 4.5.5, while [Fie-97] contains a detailed discussion. To be able to extend the limits to about 500 VBR 2-state servers/sources, a set of files is provided that uses the `long double` data type. The disadvantages of this approach consist in

- loss of portability;
- slow-down of the calculations.

The set of files provided for this case consist in

- HTML file `ffcalcl.htm`,
- CGI script `ffcoutl.cgi`,
- command-line calculation program `fflcmdln`,
- interactive calculation program `ffcl`.

The corresponding table in the appendix shows that there are merely some files being affected by this new data type.

Chapter 4

Fluid flow analysis and its implementation

This part of the report is devoted to fluid flow mathematics and how it is connected to the command-line calculation program. In the following, the corresponding source code files are given in [...].

The C++ software components are described in C++ notation, which is symbolized by typewriter typesetting. The notation of variables deviates to some extent from the notation in the theoretical part.

4.1 Objects

4.1.1 Calculation object

The main object is the calculation object of class RPMUXC that comprises the source and server objects and all the necessary settings, functions, matrices and vectors needed to carry out the fluid flow analysis. The class is defined in the header file ff_c.h and handled in ff_cmdln.cc via the pointer MPtr.

The fluid flow model is set with the function int RPMUXC::set_model(char) [ff_c_io.cc], while the setting might be read with FF_MODE RPMUXC::get_model() [ff_c.h]. FF_MODE represents an enumeration type [ff_c.h]. The function void RPMUXC::show_model() prints out information about the model.

The buffer size is set with void RPMUXC::set_Kg() [ff_c.h] and delivered by double RPMUXC::get_Kg() [ff_c.h].

4.1.2 Sources object

After the parameters for a group of sources have been read and checked to be error-free, the corresponding group is created by calling int RPMUXC::add_src(SRC_TYP typ, int n,

Parameter	Symbol	Description
typ		Either NR for CBR or NMMR for VBR 2-state sources
n	n_i^+	number of sources in group i
r0	l_i^+	"source low" bit rate for one source (NMMR only)
r1	h_i^+	"source high" bit rate for one source
l	λ_i^+	"source low" to "source high" transition rate (NMMR only)
u	μ_i^+	"source high" to "source low" transition rate (NMMR only)

Table 4.1: Parameters of the call of function RPMUXC::add_src(...) in the case of sources.

double r0, double r1, double l, double u) [ff_c_io.cc]. The parameters are given in Table 4.1. The units of the numbers are discussed in chapter 2 and in the HTML user manual. The function does not ask for the number of the group, because groups are joined successively to the source pointer array RPbaseC** RPMUXC::rbp [ff_c.h]. The class RPbaseC [ff_sbase.h] is the base class from which all kinds of sources are deducted. Therefore, common source-related functions are declared virtual: Each of the classes

- class NRPC [ff_s1r.h] — for CBR sources
- class NMMRPC [ff_s2mmr.h] — for VBR 2-state sources
- class NIRPC [ff_s2ir.h] — for VBR on-/off sources; not used in ff_cmdln
- class ooIRPC [ff_s2oor.h] — test class for a Poisson fluid arrival process; not used in ff_cmdln

implements its own version of corresponding function that is even accessible via the base-class pointer. This concept makes it possible to use different types of sources side-by-side based on the same infrastructure. The numbering of sources begins with 1, i.e. rbp[1] is the first pointer pointing to a source object. The number of source objects is limited by FF_MAX_NO_TYPES [ff_const.h].

Source parameters are shown by the function void RPMUXC::show_src(int) [ff_c_io.cc], where the parameter specifies the group of sources.

4.1.3 Servers object

Already in section 2.7.5, it has been outlined that from the fluid flow model point of view, servers are nothing else than sources with negative rates. Thus, we might use the same kind of objects both for sources and servers. However, there are some differences:

- The cell rates of servers are negative.
- The program relies on r0 $<$ r1.

Parameter	Symbol	Description
typ		Either NR for CBR or NMMR for VBR 2-state servers
n	n^-	number of servers
r0	$-h^-$	negative “server high” bit rate for one server
r1	$-l^-$	negative “source low” bit rate for one server (NMMR only)
l	μ^-	“server high” to “server low” transition rate (NMMR only)
u	λ^-	“server low” to “server high” transition rate (NMMR only)

Table 4.2: Parameters of the call of function RPMUXC::add_src(...) in the case of servers.

This is reflected in Table 4.2. Throughout the program, servers are treated as sources — with some exceptions, e.g. in load and loss probability calculations. Within the program, servers are recognized by their (absolutely seen) non-positive peak rate n*r1.

By default, function parameters that are not explained explicitly in the following should be set to 0.

4.1.4 General objects

There are some general kinds of objects being used almost everywhere in the program:

- VECTOR [ff_vect.h/cc]:
  vector of floating-point numbers, based on data type double;
- IVECTOR [ff_ivect.h/cc]:
  vector of integer numbers, mostly used for indexing purposes;
- MATRIX [ff_matr.h/cc]:
  matrix of floating-point numbers, based on data type double, which includes methods for solving linear systems of equations (output of type VECTOR);
- IMATRIX [ff_imatr.h/cc]:
  matrix of integer numbers, mostly used for indexing purposes.

Such objects might as well be used outside the fluid flow context. The only additional file they require is the file ff_const.h that contains definitions of some constants.

4.2 General functions

4.2.1 User input verification

The following functions adapt the functionality delivered by standard C functions to our specific needs:

- `check_for_int(char* str)`: checks whether `str` represents an integer (return code `0`). Floating-points and characters are not accepted (error code `1`).
- `check_for_double(char* str)`: checks whether `str` represents a floating-point number (return code `0`). Characters are not accepted (error code `1`).

4.2.2 Load calculation

Calculation of the offered load is performed by the function `double RPMUXC::At()` `[ff_c_io.cc]`. Within this function, sources and servers that are referenced via the same array of base class pointers have to be treated differently, because they contribute in different ways, c.f (2.18). Furthermore, a constant server rate `Cg` that is not used at all in `ff_cmdln`, but which might be used in `ffc`, has to be taken care of.

4.2.3 Size of the state space

The size of the state space (2.13) is automatically determined once a new source (or server) group is added. The corresponding value is returned by `int RPMUXC::get_no_states()` `[ff_c.h]`. The command-line calculation program `ff_cmdln.cc` issues a warning if the state space becomes larger than a predefined value (`WARN_NO_STATES`).

4.3 State transitions, probabilities and drift values

The state transitions are captured in the *generator matrix* $\mathbf{M}$, whose entries m_{sq} represent the transition rate from state q to state s. The generator matrix for the whole systems of sources and servers is composed out of those for groups of sources $\mathbf{M}_i^+$ and servers $\mathbf{M}^-$ by using Kronecker algebra:

$$\mathbf{M} = \mathbf{M}_1^+ \oplus \ldots \oplus \mathbf{M}_g^+ \oplus \mathbf{M}^- \tag{4.1}$$

The infinitesimal generator for a group of $n_\cdot$ CBR sources/servers is given by the neutral element of the Kronecker addition

$$\mathbf{M}_\cdot = [0] \,, \tag{4.2}$$

while for a group of $n_\cdot$ VBR 2-state sources/servers, it becomes

$$\mathbf{M}_\cdot = \begin{bmatrix} -n_\cdot\lambda_\cdot & \mu_\cdot & 0 & \cdots & 0 \\ n_\cdot\lambda_\cdot & -(n_\cdot-1)\lambda_\cdot - \mu_\cdot & 2\mu_\cdot & \cdots & 0 \\ 0 & (n_\cdot-1)\lambda_\cdot & -(n_\cdot-2)\lambda_\cdot - \mu_\cdot & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \lambda_\cdot & -n_\cdot\mu_\cdot \end{bmatrix} \tag{4.3}$$

The generator matrix determines the behavior of the state probabilities, which in the stationary case reads:

$$\mathbf{M} \cdot \vec{\pi} = 0\,. \tag{4.4}$$

The *state probability vector* $\vec{\pi}$ may also be composed from the vectors of the groups by using Kronecker algebra:

$$\vec{\pi} = \vec{\pi}_1^{+} \otimes \ldots \otimes \vec{\pi}_g^{+} \otimes \vec{\pi}^{-}\,. \tag{4.5}$$

However, we use closed formulae (2.6) and (2.14) to determine the state probabilities. If desired, the generator matrix $\mathbf{M}$ might be displayed by using the function `void RPMUXC::show_generator(MATR_OUTP matr_outp) [ff_c_io.cc]`, where the enumeration type `MATR_OUTP [ff_const.h]` determines the way how the matrix is shown.

The drift values d_s (2.16) are captured in the second matrix, the diagonal *drift matrix* $\mathbf{D}$. This matrix might be composed out of group-related *rate matrices*, which reflect the rate contribution (positive for sources, negative for servers) to each state (2.7), by Kronecker algebra:

$$\mathbf{D} = \mathbf{R}_1^{+} \oplus \ldots \oplus \mathbf{R}_g^{+} \oplus (-\mathbf{R}^{-}) \tag{4.6}$$

If a constant capacity value `RPMUXC::Cg` is specified, the right-hand side of (4.6) has to be completed with $-C\,\mathbf{I}$, where $\mathbf{I}$ is a unity matrix of suitable size. Matrix $\mathbf{D}$ might be displayed by using the function `void RPMUXC::show_drift(MATR_OUTP matr_outp)`.

The determination of the set of states with their probabilities and drift values is carried out by the function `int RPMUXC::create_all_states(int outp_en) [ff_c_io.cc]`, where `outp_en = 0` suppresses the screen output of the results. Summarized, this function creates:

- A matrix of integers `IMATRIX* RPMUXC::i_state_m` containing the indexes of the states of the groups $s_i^{+}\ \forall\, i$ and s^{-} that contribute to state s;
- The vector of state probabilities `VECTOR* RPMUXC::t_state_pr_v`;
- A matrix `MATRIX* i_crate_m [ff_c.h]` containing
  - in column 0 the drift values for each state s (row), if the constant capacity value `double RPMUXC::Cg` is zero;
  - in column i the bit rate contributions of source ($r_i^{+} > 0$) and server ($r^{-} < 0$) groups to each state s (row);
- A load vector of integers `IVECTOR* RPMUXC::load_v`, whose elements indicate which state is an
  - under-load state: element `< 0`;
  - equilibrium state: element `== 0`;
  - over-load state: element `> 0`;
- The number of overload states `int RPMUXC::no_ol_states`.

All these objects are defined within `class RPMUXC [ff_c.h]` and may be displayed by corresponding functions `[ff_c_io.cc]`.

In the case of the buffer-less fluid flow model, the state probabilities determine the probability that the buffer is full (4.32). Thus, if loss probabilities are to be calculated, we might proceed to section 4.6.7, while the probability of saturation might be determined at once (section 4.8.1).

4.4 System of differential equations

The distribution of the buffer content $\vec{F}(x)$ with

$$F_s(x) = \Pr\{X \leq x \wedge \text{state} = s\} \tag{4.7}$$

is the solution of the system of differential equations

$$\mathbf{D} \cdot \frac{\mathrm{d}}{\mathrm{d}x} \vec{F(x)} = \mathbf{M} \cdot \vec{F}(x) \tag{4.8}$$

with the boundary conditions that the buffer is never full in under-load states:

$$F_s(K) = \pi_s \quad \forall s : d_s < 0 \tag{4.9}$$

and never empty in over-load states:

$$F_s(0) = 0 \quad \forall s : d_s > 0 \tag{4.10}$$

The solution of (4.8–4.10) is given by

$$\vec{F}(x) = \sum_{q=0}^{\nu-1} a_q(K)\, \vec{\varphi}_q \exp(z_q x)\,. \tag{4.11}$$

The determination of the sets of eigenvalues $\{z_q\}$ and corresponding eigenvectors $\{\vec{\varphi}_q\}$ is discussed in the next section 4.5, while sections 4.6 and 4.7 deal with the determination of the coefficients $a_q(K)$ and $a_q(\infty)$, respectively.

4.5 Eigensystem

Transformation of (4.8) leads to an eigenvalue/-vector problem

$$z_q \mathbf{D} \cdot \vec{\varphi}_q = \mathbf{M} \cdot \vec{\varphi}_q \quad \forall q\,. \tag{4.12}$$

The set of eigenvalues $\{z_q\}$ may be obtained from the characteristic equation

$$\mathbf{det}\left[z_q \mathbf{D} - \mathbf{M}\right] = 0 \tag{4.13}$$

and the set of eigenvectors $\{\vec{\varphi}_q\}$ from solving the system of equations

$$\left[z_q \mathbf{D} - \mathbf{M}\right] \cdot \vec{\varphi}_q = \vec{0} \quad \forall q\,. \tag{4.14}$$

Unfortunately, this method implies serious numerical problems. Therefore, we use special properties of the *eigensystem* $\{\{z_q, \vec{\varphi}_q\}\}$.

type of state	d_s	z_s	entry in `load_v`
strongest negative drift	$d_0 = \min_s\{d_s\} < 0$	$:= 0$	`-2`
under-load state	$d_s < 0$	> 0	`-1`
equilibrium state	$d_s = 0$	$:= \infty$	`0`
over-load state	$d_s > 0$	< 0	`1`
strongest positive drift	$d_{\nu-1} = \max_s\{d_s\}$	$= \max_{s:z_s<0}\{z_s\}$	`2`

Table 4.3: Eigenvalue assignment, buffered model.

4.5.1 Properties of the eigenvalues

The following properties are very important for the treatment of the eigenvalues.

- Due to the time-reversability of the processes under consideration, the eigenvalues are real.
- There is always an eigenvalue 0.
- There are as many negative eigenvalues as there are over-load states.
- The number of positive eigenvalues is given by the number of under-load states minus one.
- Originally, the analysis could not be carried out if equilibrium states occurred [Ani-82]. However, we are able to treat equilibrium states in the same way as over- or under-load states due to
$$\lim_{d_s \to \pm 0} z_s = \mp\infty . \tag{4.15}$$
There are as many "unspecified" eigenvalues as there are equilibrium states.

Due to the latter properties, we may assign the eigenvalues to states at least in a formal way as shown by Table 4.3. After calculation, the eigenvalues are stored in `VECTOR* RPMUXC::eval_v`. For this assignment, it is very important that state 0 exhibits the strongest negative and state $\nu - 1$ the strongest positive drift. Therefore, `r0 < r1` has to be guaranteed when creating the source objects. In the program, eigenvalues are marked as being "∞" by explicitly setting (":=") them to `FF_D_B_OVER [ff_const.h]`. Section 4.7 will reveal that for *infinite buffer*, only the negative eigenvalues are needed; in this case, the program sets the positive eigenvalues also to "∞" and thus skips the calculation of the corresponding eigenvectors.

Making use of a generating functions approach for the eigenvectors, see [Ani-82, Kos-84, Ste-91, Fie-97] *et al*, the other eigenvalues might be obtained from formula whose order is given by the two times the number of VBR 2-state groups of servers and sources. Carrying out this analysis is only feasible for *one* such group, as one group of VBR 2-state sources in combination with one group of VBR 2-state servers already leads to equations of order four to be solved. Even though a closed solution exists, its implementation is much too complicated.

The analysis of two (or more) groups of VBR 2-state sources in combination with one group of VBR 2-state servers needs to be carried out numerically anyway, and way to do this will be described in the following subsections.

The determination of eigenvalues is performed by the functions `int RPMUXC::eigensys(int chk_en, int ev_out_en, int outp_en)` and `int RPMUXC:: eval_srch(int i, int chk_en, int trace_en, int outp_en) [ff_c_es.cc]`. The only parameter that is no control parameter is `i` which denotes the current state.

4.5.2 The inverse eigenvalue problem

The *inverse eigenvalue problem* to (4.12) reads

$$\gamma_q(z_q)\,\vec{\varphi}_q = \left(\mathbf{R} - \frac{1}{z_q}\,\mathbf{M}\right)\cdot\vec{\varphi}_q \tag{4.16}$$

with

$$\gamma_q(z_q) = \sum_{i=1}^{g} \gamma_{q,i}^{+}(z_q) + \gamma_q^{-}(z_q) = 0\,. \tag{4.17}$$

The latter formula might be used to determine the eigenvalues z_q numerically due to the fact that the formulae for $\gamma_q(z_q)$ are given in closed form. Observe that the value 0 on the right-hand-side of (4.17) has to be replaced by a constant capacity value C if `RPMUXC::Cg` should be used.

The *eigenvalues of the inverse problem* are given

- for a group of CBR sources as

$$\gamma_{q,i}^{+}(z_q) \;=\; h_i^{+} \tag{4.18}$$

- for a group of VBR 2-state sources, out of which k sources are "high" in state q, as

$$\begin{aligned}\gamma_{q,i}^{+}(z_q) \;=\; & n_i^{+} l_i^{+} + \frac{1}{2z_q}\Bigg(n_i^{+}\left((z_q h_i^{+} - z_q l_i^{+}) + \lambda_i^{+} + \mu_i^{+}\right) + \\ & + \operatorname{sign}(z_q)\,(2k - n_i^{+})\sqrt{4\lambda_i^{+}\mu_i^{+} + (z_q h_i^{+} - z_q l_i^{+} - \lambda_i^{+} + \mu_i^{+})^2}\Bigg)\end{aligned} \tag{4.19}$$

- for a group of CBR servers as

$$\gamma_q^{-}(z_q) \;=\; -h^{-} \tag{4.20}$$

- for a group of VBR 2-state servers, out of which k servers are "high" in state q, as

$$\begin{aligned}\gamma_q^{-}(z_q) \;=\; & -n^{-} l^{-} - \frac{1}{2z_q}\Bigg(n^{-}\left((z_q h^{-} - z_q l^{-}) + \lambda^{-} + \mu^{-}\right) + \\ & + \operatorname{sign}(z_q)\,(2k - n^{-})\sqrt{4\lambda^{-}\mu^{-} + (z_q h^{-} - z_q l^{-} - \lambda^{-} + \mu^{-})^2}\Bigg)\end{aligned} \tag{4.21}$$

For sources and servers, the corresponding eigenvalues (4.18) and (4.20) or (4.19) and (4.21) differ by their sign. However, this is happening automatically when we assign the source object parameters as described in section 4.1.3. These eigenvalues are implemented as `virtual inline` functions

- for CBR sources or servers in `double NRPC::cakt() [ff_s1r.cc]`
- for VBR 2-state sources or servers in `double NMMRPC::cakt() [ff_s2mmr.cc]`

which may be called via the source pointer array `rbp`. Before doing that, the number of "high" sources has to be set in the source objects via the function `int RPbaseC::set_a_state(int k_inp) [ff_sbase.h]`; the same applies for the eigenvalue (`void RPbaseC::set_a_eval(double z_inp) [ff_sbase.h]`).

4.5.3 Numerical search for eigenvalues

Based on the eigenvalues of the inverse problem, the search for an eigenvalue z_q is carried out by `int RPMUXC::eval_srch(int i, int chk_en, int trace_en, int outp_en) [ff_c_es.cc]` with a relative deviation defined by `FF_ZS_TOL`, usually 10^{-15}. For `trace_en = 1`, the search process might be observed. The sign function in (4.19) and (4.21) is not a problem because the sign of the eigenvalue is known in advance: Eigenvalues that are assigned to under-load states are searched between 0 and $+\infty$, while eigenvalues that are assigned to over-load states are searched between $-\infty$ and 0. The search itself is interval-based: First, an interval $]z_l, z_u[$ is determined that includes z_q, which is the case if the left-hand-side of (4.17) has different sign for z_l and z_u, respectively. The interval is halved and the sub-interval is chosen that contains z_q. This procedure is continued until the desired precision has been reached.

4.5.4 Determination of the eigenvectors

Once an eigenvalue has been calculated, the corresponding eigenvector may be computed, which is composed of sub-eigenvectors for each group

$$\vec{\varphi}_q = \vec{\varphi}^+_{q,1} \otimes \ldots \otimes \vec{\varphi}^+_{q,g} \otimes \vec{\varphi}^-_q \,. \tag{4.22}$$

All the eigenvectors are normalized in the following way:

$$\sum_s \varphi_{q,s} = 1 \tag{4.23}$$

The eigenvector for $z_0 = 0$ is given by the vector of the state probabilities

$$\vec{\varphi}_0 = \vec{\pi}\,, \tag{4.24}$$

while in the limit $d_q \to 0$, the eigenvector that comes along with indeterminate eigenvalues (equilibrium states) becomes a unit vector in q-direction:

$$\lim_{z_q \to \infty} \vec{\varphi}_q = \vec{e}_q \tag{4.25}$$

Depending on the type of sources or servers, the eigenvectors are given as follows:

- For a CBR source or server group, we obtain

$$\vec{\varphi}_q = [1] \tag{4.26}$$

  which is implemented in the virtual function `int NRPC::evaluate_evec()`.

- For a VBR 2-state source or server group, we have to use the eigenvector-generating function approach described in [Ani-82, Fie-97]. Together with the two residua that look little different for sources and servers,

$$\begin{aligned} \mathrm{res}^{+}_{q,1/2}(z_q) &= \frac{1}{2\lambda^+}\left(\lambda^+ - \mu^+ - z_q(h^+ - l^+) \pm \right. \\ &\quad \left. \pm\sqrt{4\lambda^+\mu^+ + (\lambda^+ - \mu^+ - z_q(h^+ - l^+))^2}\right) \end{aligned} \tag{4.27}$$

$$\begin{aligned} \mathrm{res}^{-}_{q,1/2}(z_q) &= \frac{1}{2\mu^-}\left(\mu^- - \lambda^- + z_q(h^- - l^-) \pm \right. \\ &\quad \left. \pm\sqrt{4\lambda^-\mu^- + (\mu^- - \lambda^- + z_q(h^- - l^-))^2}\right) \end{aligned} \tag{4.28}$$

  and with $q^{\pm}$ sources/servers out of $n^{\pm}$ being in the "high" state in state q, the components k of the sub-eigenvectors are given in closed form:

$$\varphi^{\pm}_{q,k} = (-1)^{n^{\pm}-k} \sum_{\xi=0}^{q^{\pm}} \binom{q^{\pm}}{\xi} \binom{n^{\pm}-q^{\pm}}{k-\xi} \left(\mathrm{res}^{\pm}_{q,1}(z_q)\right)^{q^{\pm}-\xi} \left(\mathrm{res}^{\pm}_{q,2}(z_q)\right)^{n^{\pm}-q^{\pm}-k+\xi} \tag{4.29}$$

  However, formula (4.29) is implemented in `int NMMRPC::evaluate_evec()` in a modified way that is described in [Fie-97].

After having been computed for each group separately, the sub-eigenvectors are joined together (4.22) by the function `int RPMUXC::evaluate_a_evec(int chk_en, int part_en, int outp_en)` `[ff_c_es.cc]`. The eigenvectors are stored in the matrix `MATRIX* RPMUXC::evec_m` `[ff_c.h]`.

4.5.5 Numerical problems

For large groups, the computation of (4.29) is numerically critical: The binomial coefficients may lead to numerical overflow, while the powers of the residua may lead to numerical underflow. This problem may be solved in two ways:

1. By calculating the product in (4.29) in a logarithmic way `[ff_s2mmr.cc]`; this option is chosen automatically per group if the its number of sources/servers exceeds `FF_EVEC_N_B` `[ff_const.h]` or the probability that one source/server is in the "high" state becomes less than `FF_EVEC_N_B [ff_const.h]`. This *logarithmic calculation* is much slower than the standard calculation (about factor 10).

z_q	$a_q(K)$	Remarks
$=0$	>1	
>0	>0	
$\to \pm\infty$	$\to 0^{\pm}$	in the limit
<0	<0	

Table 4.4: Properties of the coefficients for finite buffer.

2. By calculating the product in (4.29) with help of `long double` variables `[fffls2mmr.cc]`. This leads to a drastic improvement of the numerical stability of the whole fluid flow analysis, but `long double` variables may not be used with each C++ compiler. Therefore, a separate source code file `fffls2mmr.cc` implements this case. This *long double calculation* is even much slower (about factor 100 compared to the standard calculation); therefore, it should only be used if the number of sources/servers in one group exceeds 200, which is of course no hard limit, but also influenced by $\alpha_i^{\pm}$.

The difference in terms of stability are shown in [Fie-97].

4.6 Coefficients for finite buffer

After the eigenvalues and eigenvectors have been determined, the coefficients of the solution have to be calculated by solving the linear system of equations given by (4.9) and (4.10).

4.6.1 Properties of the coefficients

Some properties of the coefficients belonging to eigenvalues z_q and eigenvectors $\vec{\varphi}_q$ are listed in Table 4.4. The positive coefficients that belong to the "unstable" eigenvalues force (4.9). Furthermore, and together with (4.25), it becomes obvious that *equilibrium states do not contribute to the solution at all*.

4.6.2 Deletion of critical states

Due to (4.15), states with almost vanishing drift may cause

1. numerical underflow, if the drift is slightly negative;
2. numerical overflow, if the drift is slightly positive.

The first problem *may*, but the second problem *does* affect the solution procedure — the numerical calculations collapse. From Table 4.4, we know that the contribution of states with vanishing drift becomes arbitrarily small. Therefore, we mark all states that may not be used due to

1. $\exp(z_q K) <$ `FF_EZK_OL` (typically 10^{-300}) for over-load states,

2. $\exp(z_q K) >$ `FF_EZK_UL` (typically 10^{+300}) for under-load states.

This happens by setting the corresponding element of `IVECTOR* RPMUXC::eq_used [ff_c.h]` to `0`. Equilibrium states are marked the same way by the function `int RPMUXC::delete_worst_states(int outp_en, double ezk_ol, double ezk_ul) [ff_c_eq.cc]`. The number of used equations is contained in `int RPMUXC::no_used [ff_c.h]`.

4.6.3 The system of equations

The system of equations (4.9, 4.10) is put together by the function `int RPMUXC::create_equ_loss(int out) [ff_c_eq.cc]` from

- the matrix of eigenvectors `evec_m`,
- the vector of eigenvalues `eval_v`,
- the buffer size `Kg`,
- the vector of state probabilities `t_state_pr_v`

based on the information contained in the flag vector `eq_used`: Equation *s* is created only, if element *s* of `eq_used` is not zero. The system is stored in `MATRIX* RPMUXC::eq_syst [ff_c.h]` that has `no_used` rows (= equations) and `no_used + 1` columns, where the last column contains the right-hand sides of (4.9f).

4.6.4 Solution of the system

The solution of the system of equations is carried out by the function `int RPMUXC::solve_eqsyst(int outp_en)`. This function operates mainly on the matrix `eq_syst` by using the function `VECTOR* MATRIX::solve(...) [ff_matr.cc]`, thereby changing `eq_syst`! By default, the *Gauss elimination method with complete pivoting* is set as solution procedure (c.f `#define FF_SOLV_M GAUSS_T [ff_const.h]`), because it has shown up to be the numerically most versatile of all the methods implemented in `ff_matr.cc` [Fie-97, Fie-99]. The result is stored in the `VECTOR* RPMUXC::solut_v [ff_c.h]`, which in most cases merely contains the subset of non-vanishing coefficients due to the fact that some equations had to be left out, c.f subsection 4.6.2.

4.6.5 Assignment of the coefficients

With the flag vector `eq_used`, the coefficient vector `VECTOR* RPMUXC::coeff_v [ff_c.h]` is created from the solution vector `solut_v`, which is performed by the function `int RPMUXC::solut_2_coeff_v(int outp_en) [ff_c_eq.cc]`. If element s of `eq_used` had been marked with 0, then coefficient s of `coeff_v` will also be set to zero.

4.6.6 Regeneration of the flag vector

If the buffer size is changed, then the procedure has to be carried out from step 4.6.2; a reconstruction of the eigensystem is not necessary. The flag vector `eq_used` has to be prepared for new marking of critical states by calling the function `int RPMUXC::regen_eq_used() [ff_c_es.cc]`.

4.6.7 The probabilities of full buffer

After the coefficients $a_q(K)$ have been determined, the probabilities that the buffer gets full in over-load state s are given by

$$u_s^+ = \pi_s - \lim_{x\to K^-} F_s(x) \tag{4.30}$$

$$= \pi_s - \sum_{q=0}^{\nu-1} a_q(K)\,\vec{\varphi}_q \exp(z_q K)\,. \tag{4.31}$$

For the buffer-less model, this reduces to

$$u_s^+ = \pi_s\,. \tag{4.32}$$

Both cases are treated by the function `int RPMUXC::create_full_v(int outp_en) [ff_c_xd.cc]` that writes the result to `VECTOR* RPMUXC::full_v [ff_c.h]`.

4.7 Coefficients for infinite buffer

For infinite buffer, the linear system of equations looks different. The condition (4.9) now reads

$$\lim_{x\to\infty} F_s(x) = \pi_s \quad \forall\, s : d_s < 0 \tag{4.33}$$

This has special consequences for the coefficients that are described in the next subsection.

z_q	$a_q(K)$	Remarks
$=0$	$=1$	
>0	$=0$	
$\to -\infty$	$\to 0^-$	in the limit
<0	<0	

Table 4.5: Properties of the coefficients for finite buffer.

4.7.1 Properties of the coefficients

Compared to the buffered case (see Table 4.4), the properties of the coefficients have changed (Table 4.5). The positive coefficients that belong to the "unstable" eigenvalues have to become zero. This in turn leads to the coefficient $a_0 = 1$. The only coefficients that have to be determined are those for the over-load states; the corresponding system of equations reads

$$\sum_{q:d_q>0} a_q(\infty)\,\varphi_{q,s} = -\pi_s \quad \forall s : d_s > 0\,. \tag{4.34}$$

For only one group of VBR 2-state sources or servers, [Ani-82] presented a very elegant, closed-form solution which unfortunately cannot be translated to the general case.

4.7.2 Deletion of critical states

As the equations (4.33) for the under-load states are not needed due to the vanishing coefficients, we cannot get numerical over- or underflow from exponential functions containing the eigenvalues. Thus, the function `int RPMUXC::delete_worst_states(int outp_en, double ezk_ol, double ezk_ul) [ff_c_eq.cc]` is not necessary for infinite buffer.

4.7.3 The system of equations

The system of equations (4.10) is composed by the function `int RPMUXC::create_equ_delay(int out) [ff_c_eq.cc]` from

- the matrix of eigenvectors `evec_m`,
- the vector of eigenvalues `eval_v`,
- the vector of state probabilities `t_state_pr_v`

and is stored in `MATRIX* RPMUXC::eq_syst [ff_c.h]` that has `no_ol_states` rows (= equations) and `no_ol_states + 1` columns. Again, the flag vector `eq_used` is used to indicate which states contribute.

4.7.4 Solution of the system

The solution of the system of equations is carried out by the same function as in the finite buffer case. As the system of equations in the infinite buffer case is smaller than in the buffered case, the solution needs less time, whereas the numerical stability is about the same [Fie-97, Fie-99]. Again, the result is stored in the vector `solut_v`.

4.7.5 Assignment of the coefficients

The coefficient vector `coeff_v` is produced from `solut_v` based on `eq_used` as described before, with the only difference that element zero is set to one.

4.8 Quality of Service-related parameters

4.8.1 Probability of saturation

The *probability of saturation* (2.20) are obtained from the function `double RPMUXC::sat_prob(int outp_en) [ff_c_go.c]`. The function uses the vector of state probabilities `t_state_pr_v` and the flag vector `load_v`.

4.8.2 Loss probability

Based on the vector of probabilities that the buffer is full in different states, `full_v`, the function `double RPMUXC::loss_prob(int type, int part_en, int inp_rep, int outp_en) [ff_c_go.cc]` delivers

- the *individual loss probability* (2.21) for a certain group of sources by specifying `type =` number of the group;
- the *total loss probability* (2.22) by specifying `type = 0`;
- an error message and the value 0, if `type` refers to a server group.

This function uses the drift/rate matrix `i_crate_m` and the flag vector `load_v`.

4.8.3 Overflow probability of a buffer threshold

From (4.11) and (4.23), we obtain the complementary distribution function of the buffer content

- for a finite buffer ($0 \leq x \leq K$) as

$$G(x) = 1 - \sum_{q=0}^{\nu-1} a_q(K) \exp(z_q x)\,; \tag{4.35}$$

- for an infinite buffer together with Table 4.5 as

$$G(x) = - \sum_{q \in \mathsf{S}^o} a_q(\infty) \exp(z_q x)\,. \tag{4.36}$$

This is carried out by the function `double RPMUXC::compl_distrib_total(double x)` `[ff_c_xd.h]` that needs the vector of eigenvalues `eval_v` and the coefficient vector `coeff_v`. Observe that the function does not perform any check on the value `x` for being reasonable.

Chapter 5

Results

5.1 Channel fading

The first example considers one channel in a mobile environment that becomes partly unavailable due to channel fading or the correction of errors. We assume that due to such unexpected error conditions, the channel bit rate sinks temporarily to merely 50 % of its original bit rate, which is the same than that of the source when transmitting ($l^- = 0.5h^{+/-}$). The source has on-/off characteristics with a low activity factor of $\alpha^+ = 0.01$ and a certain mean cycle time τ^+, while the probability that the channel (= server) has its full capacity may vary between 0.5 and 0.9 and the mean cycle time of the channel may deviate from that of the source by several orders of magnitude. The Quality of Service (QoS)-related parameter is the loss probability. More information can be obtained from a conference paper [Fie-00b] that is one of the basic results from this project.

We shall now look at some results. The figures have been produced using gnuplot's eepic interface and modified afterwards. Figure 5.1 demonstrates which influence the ratio of the mean cycle times of servers and sources has on loss probabilities. Especially when the probability that the server is fully available is quite high, the loss probability varies by about eight (!) orders of magnitude between 10^{-10} and 10^{-2}! A low loss probability is obtained if the server's cycle time is much shorter than that of the source. However, if the mean cycle time of the server becomes equal or larger than that of the source, the loss probability reaches quite high values. For QoS, a slowly varying channel capacity with quite long "channel low" times represents a danger which should not be underestimated.

How do we cope with this problem? One possibility consists of adapting the buffer size in a way that a desired loss probability is not surpassed. Figure 5.2 shows the required buffer size for a loss probability of 10^{-6} as a function of the mean cycle time ratio for the case $\alpha^- = 0.9$. Compared to a fast-varying channel, a slow-varying channel implies a need of up to ten times as much buffer space.

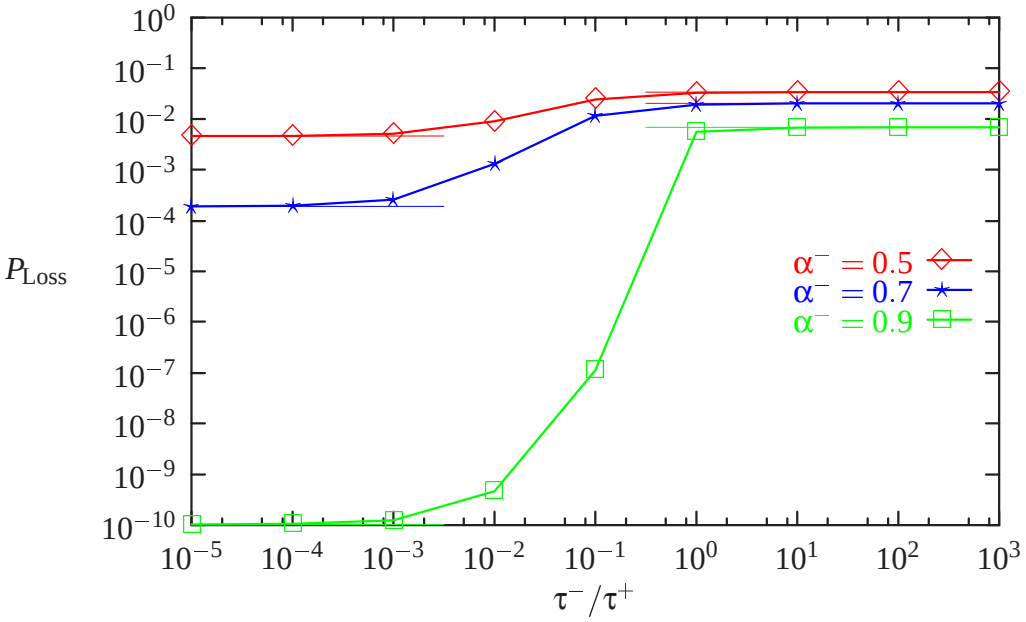

Figure 5.1: Loss probability versus the mean cycle time ratio (buffer size = mean burst size).

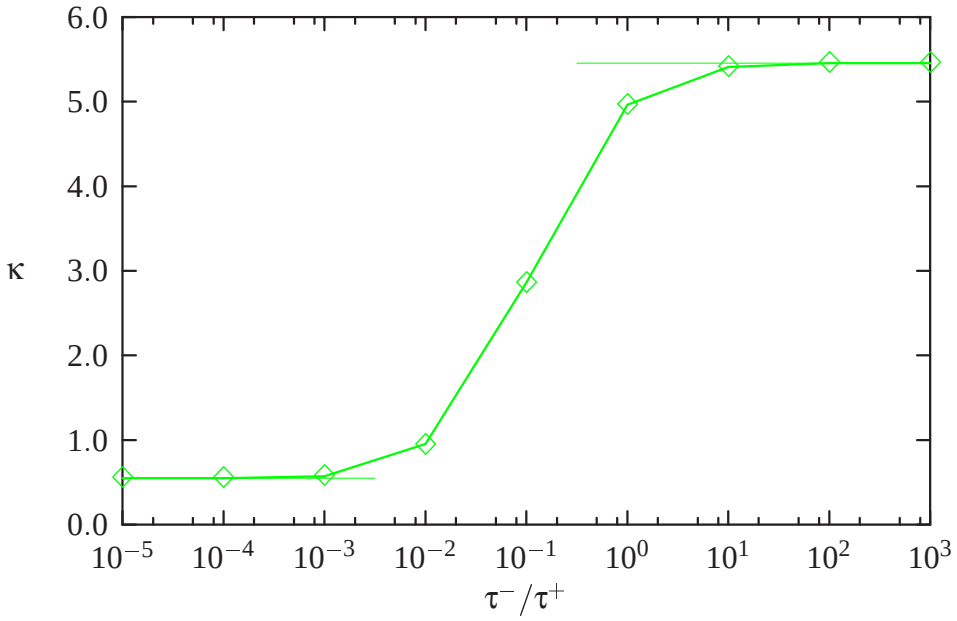

Figure 5.2: Required buffer size (to achieve a loss probability of 10^{-6}) versus the mean cycle time ratio, server high-activity factor $\alpha^{-} = 0.9$.

α^-	τ^-/τ^+	G(15 mean bursts)		
		$n^+ = 50$	$n^+ = 100$	$n^+ = 150$
1.000	—	1.55×10^{-6}	1.60×10^{-7}	2.03×10^{-8}
0.999	1	1.68×10^{-6}	1.78×10^{-7}	2.31×10^{-8}
0.999	10	1.77×10^{-6}	2.60×10^{-7}	6.64×10^{-7}
0.999	100	3.61×10^{-4}	1.26×10^{-3}	1.97×10^{-3}

Table 5.1: Overflow probability of buffer threshold = 15 $\times$ mean burst length.

5.2 Link breakdown

The second example looks at the following scenario: An ATM link carries n^+ virtual paths (VP) with on-/off behavior (acitvity factor $\alpha^+ = 0.4$) and peak bit rate $h^+ = 1$ Mbit/s. The VPs may share the full link capacity among themselves. The offered load is fixed to $A = 0.8$, so that the capacity becomes $\frac{n^+}{2}$ Mbit/s. Considering peak rate allocation, the link (= server) is over-booked by 100 %.

We shall study the effect of link breakdown on the buffer content distribution $G(x)$. We assume that the availability is lowered from $\alpha^- = 100$ % to $\alpha^- = 99.9$ %, thus raising the offered load slightly to 0.8008. The investigations are carried out for different numbers of connections n^+ and, as before, for different mean cycle time ratios between servers and sources τ^-/τ^+.

Table 5.1 shows some quite interesting results, which consist in overflow probabilities of a buffer threshold size of 15 mean bursts for different numbers of VPs and different link capacities. If the link is fully available, we observe that queuing becomes less as the number of VPs rises. Such a behavior reflects multiplexing gain that becomes the larger, the more connections share a common link. If the link's availability sinks to 99.9 % and the mean cycle times of sources and servers are matched, the changes of the overflow probability are marginal, as expected. However, if the mean cycle time of the link rises in comparison to that of the sources, i.e. if the link changes more slowly between "up" and "down", queuing becomes much more critical — and worse for larger numbers of VPs. Already for a mean cycle time of the link that is ten times as long as that for one source, the trend that the QoS becomes better for a larger number of VPs — as known from the 100 % = CBR link case – does not hold anymore. If we raise the mean cycle time of the link once more by factor ten, this trend is completely inverted!

We may summarize our observations as follows: For links that change their state much slower than VPs,

1. heavy queuing occurs;
2. the multiplexing gain sinks if the number of connections rises.

While the first result is underlined by figures 5.3 and 5.4 (unchanged PostScript output from the program), the last result may have significant impact on the dimensioning of systems if reliability comes into the game.

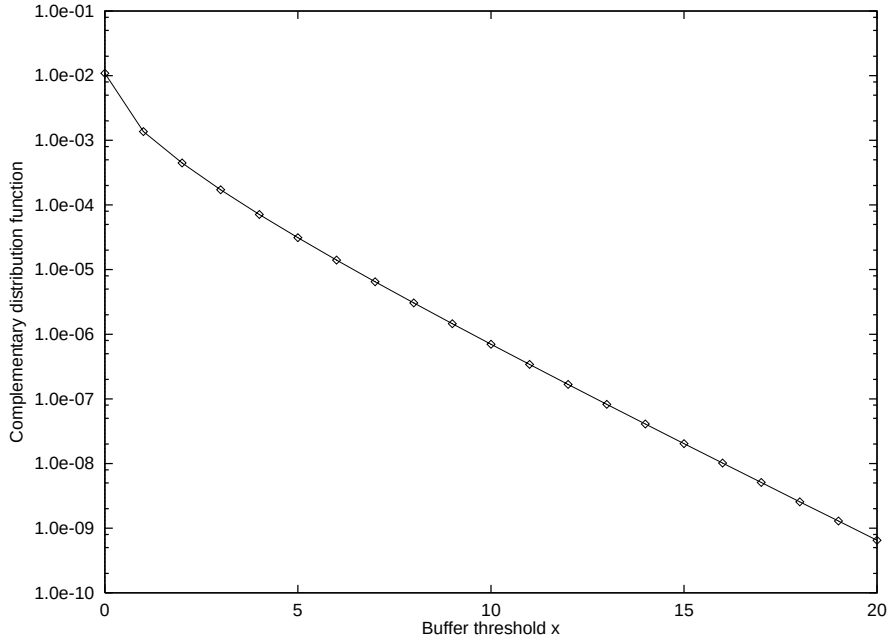

Figure 5.3: Overflow probabilities versus buffer threshold size in mean burst lengths for 150 connections and 100 % link availability.

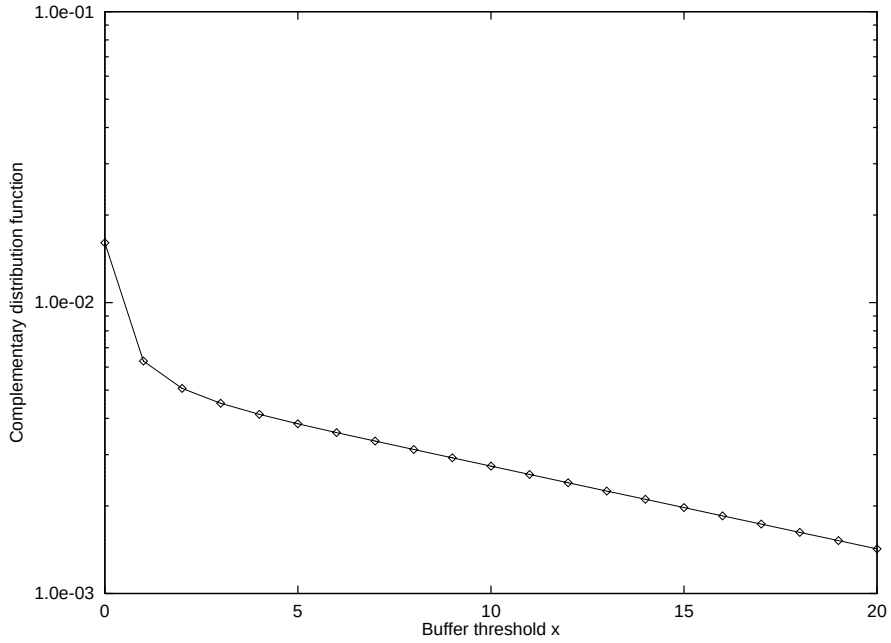

Figure 5.4: Overflow probabilities versus buffer threshold size in mean burst lengths for 150 connections and 99.9 % link availability, mean cycle time of the link = 100 mean cycle times of a VP.

Chapter 6

Summary and outlook

In this project, the fluid flow model has been applied successfully to model variable server rates, as they appear in mobile environments at the edge of the networks, but also within the core network itself.

A user-friendly, contemporary and flexible computing environment has been created that is based on a web/compute server concept, thus providing a well-known graphical user interface and multi-user facilities. The same calculation program might be used via the graphical user interface and for batch processing. Great effort has been put into the numerical stabilization of the calculations. The fluid flow analysis has been described together with its implementation, thus enabling programmers to get a better idea on how analysis and implementation get together.

Two case studies reveal how important the consideration of a variable server rate is and how dangerous a slowly-varying server capacity might become for Quality of Service (QoS). In the latter case, even the well-known “law” that multiplexing gain is rising together with the number of connections does not hold anymore.

The case studies reveal a strong dependency between results and system parameters. Here, parameters of realistic systems should be used to study the problems of varying server rates for practical systems. As the fluid flow model presents a unifying framework for network-wide QoS, combinations of network elements and links should be considered as well. More complicated source and server models than the VBR-2 state model with exponentially distributed phases should be taken into account, which implies the need to develop numerically stabilized algorithms. The list of scenarios presented in the introduction gives only a vague notion about the very many problems that the fluid flow model might be applied to in the future. Let’s continue!

Bibliography

[Ani-82] D. Anick, D. Mitra and M. M. Sondhi. Stochastic theory of a data-handling system with multiple sources. *The Bell System Technical Journal*, 61(8):1871–1894 (1982).

[Elw-93] A.I. Elwalid and D. Mitra. Effective bandwitdh of general Markovian traffic sources and admission control of high speed networks. *IEEE/ACM Transactions on Networking*, 1(3):329–343 (1993).

[Elw-94] A.I. Elwalid and D. Mitra. Statistical multiplexing with loss priorities in rate-based congestion control of high-speed networks. *IEEE Transactions on Communications*, 42(12):2989–3002 (1994).

[Fie-97] M. Fiedler and H. Voos. *Fluid flow-Modellierung von ATM-Multiplexern. Mathematische Grundlagen und numerische Lösungsmethoden.* München: Utz, 1997, ISBN 3-89675-251-0.

[Fie-98] M. Fiedler and H. Voos. *Erforderliche Kapazität beim Multiplexen von ATM-Verbindungen.* Ph.D. thesis, Universität des Saarlandes, Saarbrücken 1998. München: Utz, 1998, ISBN 3-89675-385-1.

[Fie-99] M. Fiedler and H. Voos. How to win the numerical battle against the finite buffer stochastic fluid flow model. *COST 257 temporary document 257TD(99)38.*

[Fie-00a] M. Fiedler and U.R. Krieger. The fluid flow model with variable server capacity. *COST 257 temporary document 257TD(00)18.*

[Fie-00b] M. Fiedler and U.R. Krieger. *The impact of varying channel capacity on the quality of advanced data services in PCS networks. To appear at 12th ITC Specialist Seminar on Mobile Systems and Mobility, Lillehammer, March 22-24, 2000.*

[Gué-91] R. Guérin, H. Ahmadi and M. Naghshineh. Equivalent capacity and its application to bandwidth allocation in high-speed networks. *IEEE Journal on Selected Areas in Communications*, 9(7):968–981 (1991).

[Kim-98] J.G. Kim and M. Krunz. Effective bandwith in wireless ATM networks. *Proc. MOBICOM 98*, Dallas, 1998, pp 233–241.

[Kos-74] L. Kosten. Stochastic theory of a multi–entry buffer. *Delft Progress Report, Series F*, 1:10–18 (1974).

[Kos-84] L. Kosten. Stochastic theory of data handling systems with groups of multiple sources. *Proc. of IFIP WG 7.3 – TC 6 International Symposium on the Performance of Computer Communication Systems*, Zürich (1984) pp 321–331.

[Kon-94] K.P. Kontovasilis and N.M. Mitrou. Bursty traffic modelling and efficient analysis algorithms via fluid flow models for ATM IBCN. *Annals of Operations Research*, 49:279–323 (1994).

[Kro-99] D.P. Kroese and V.F. Nicola. Efficient estimation of overflow probabilities in queues with breakdowns. *Performance Evaluation*, 36–37:471–484 (1999).

[Lam-97] C.H. Lam and T.T. Lee. Fluid flow models with state-dependent service rate. *Commun. Statist. – Stochastic Models*, 13(3):547–576 (1997).

[Mit-88] D. Mitra. Stochastic theory of a fluid flow model of producers and consumers coupled by a buffer. *Advances in Applied Probability*, 20:646–676 (1988).

[Nag-91] R. Nagarajan, J.F. Kurose and D. Towsley. Approximations techniques for computing packet loss in finite-buffered voice multiplexers. *IEEE JSAC*, 9(3):368–377 (1991).

[Ram-99] V. Ramaswami. Matrix analytic methods for stochastic fluid flows. *Proc. ITC-16*, Edinburgh (1999) pp 1019–1030.

[Sch-98] W.R.W. Scheinhardt: *Markov-modulated and Feedback Fluid Queues*. Ph.D. thesis, University of Twente, The Netherlands (1998).

[Ste-91] T. Stern and A. Elwalid. Analysis of separable markov-modulated rate models for information-handling systems. *Advances in Applied Probability*, 23:105–139 (1991).

[Tuc-88] R. Tucker. Accurate method for analysis of a packet-speech multiplexer with limited delay. *IEEE Transactions on Communications*, 36(4):479–483 (1988).

[Tur-93] W. Turin and M.M. Sondhi. Modeling error sources in digital channels. *IEEE JSAC*, 11(3):340–347 (1993).

[Yan-95] T. Yang and D.H.K. Tsang. A novel approach to estimating the cell loss probability in an ATM multiplexer loaded with homogeneous on-off sources. *IEEE Transactions on Communications*, 43(1):117–126 (1995).

Appendix

1. List of files for building the executables
2. Installation
3. Configuration
4. Web pages and user manual

List of files for building the executables

code file	executable: standard version `ff_cmdln`	executable: standard version `ffc`	executable: long double vers. `fflcmdln`	executable: long double vers. `ffcl`	Remarks
`ff_c.h`	√	√	√	√	`calculation class RPMUXC`
`ff_const.h`	√	√	√	√	definition of constants
`ff_imatr.h`	√	√	√	√	integer (= index) matrix `class IMATRIX`
`ff_ivect.h`	√	√	√	√	integer (= index) vector `class IVECTOR`
`ff_matr.h`	√	√	√	√	matrix `class MATRIX`[1]
`ff_vect.h`	√	√	√	√	vector `class VECTOR`
`ff_sbase.h`	√	√	√	√	source/server base `class RPbaseC`
`ff_s1r.h`	√	√	√	√	CBR source/server `class NRPC`
`ff_s2ir.h`	√	√	√	√	VBR on-/off source/server `class NIRPC`
`ff_s2mmr.h`	√	√	√	√	VBR high-/low source/server `class NMMRPC`
`ff_s2oor.h`	√	√	√	√	VBR Poisson test `class ooIRPC`
`ff_c_io.cc`	√	√	√	√	`RPMUXC` – I/O routines
`ff_c_es.cc`	√	√	√	√	`RPMUXC` – state space and eigensystem
`ff_c_eq.cc`	√	√			`RPMUXC` – build system of equations
`fflc_eq.cc`			√	√	`RPMUXC` – build system of equations
`ff_c_so.cc`	√	√	√	√	`RPMUXC` – solve system of equations
`ff_c_xd.cc`	√	√	√	√	`RPMUXC` – distribution of buffer content
`ff_c_go.cc`	√	√	√	√	`RPMUXC` – Grade/Quality of Service
`ff_c_cr.cc`		√		√	`RPMUXC` – required capacity[2]
`ff_c_cn.cc`		√		√	`RPMUXC` – proportional required capacity[2]
`ff_c_ca.cc`		√		√	`RPMUXC` – CAC regions
`ff_cmdln.cc`	√		√		`main program for command-line version`
`ff_main.cc`		√			`main program for interactive version`
`fflmain.cc`				√	`main program for interactive version`
`ff_imatr.cc`	√	√	√	√	integer (= index) matrix `class IMATRIX`
`ff_ivect.cc`	√	√	√	√	integer (= index) vector `class IVECTOR`
`ff_matr.cc`	√	√	√	√	matrix `class MATRIX`
`ff_vect.cc`	√	√	√	√	vector `class VECTOR`
`ff_sbase.cc`	√	√	√	√	source/server base `class RPbaseC`
`ff_s1r.cc`	√	√	√	√	CBR source/server `class NRPC`
`ff_s2ir.cc`	√	√			VBR on-/off source/server `class NIRPC`
`ffls2ir.cc`			√	√	VBR on-/off source/server `class NIRPC`
`ff_s2mmr.cc`	√	√			VBR high-/low source/server `class NMMRPC`
`ffls2mmr.cc`			√	√	VBR high-/low source/server `class NMMRPC`
`ff_s2oor.cc`	√	√			VBR Poisson test `class ooIRPC`
`ffls2oor.cc`			√	√	VBR Poisson test `class ooIRPC`
`makefile`	√[3]	√[3]	√[4]	√[4]	utility for compilation

[1] includes methods for solving linear systems of equations
[2] for notions, see [Fie-98]
[3] calls gnu compiler `g++`
[4] calls Sun compiler `CC`, links in `sunmath` library

Installation

The web/compute server requires the following files in the *WWW_home_directory* (e.g. `www` or `wwwroot`) that is to be found under `http://`*`Server_address.WWW_home`*:

- the HTML forms `ffcalc.htm` and `ffcalcl.htm`[1]
- the CGI script `ffcout.cgi` and `ffcoutl.cgi`[1]
- the executables `ff_cmdln(.exe)` and `fflcmdln(.exe)`[1]
- the Gnuplot executable
  - UNIX/LINUX: `gnuplot`
  - Windows 95/98/NT: `wgnupl32.exe`
- in the directory `man`, the HTML manual and help files
  1. `index.htm` (root file)
  2. `usage.htm`
  3. `units.htm`
  4. `qosthres.htm`
  5. `sources.htm`
  6. `servers.htm`
  7. `output.htm`
  8. `excelgnu.htm`
  9. `errcodes.htm`
  10. `load.htm`
  11. `warnings.htm`
  12. `nostates.htm`
  13. `strange.htm`
  14. `software.htm`
  15. `param.htm`
  16. `websconf.htm`
  17. `securadm.htm`

[1] only, if the long double version of the calculation program is available

Configuration

The following configuration settings might be necessary:

- *Apache server*: in file `access.conf`,
  - set permission to execute CGI: add
    ```
    <Directory/
    Options FollowsSymLinks ExecCGI AllowOverride None
    </Directory>
    ```
  - allow directory: add/modify
    ```
    <Directory "WWW_home_directory">
    Options ExecCGI Includes
    AllowOverride None
    </Directory>
    ```
  - allow hosts to contact: add
    ```
    order allow,deny
    allow from host
    ```

  in file `httpd.conf`,

  - set suitable time-out: modify
    `Timeout max. calculation time`

- *Enable PostScript*: in file in `ffcout.cgi` and `ffcoutl.cgi`[1],
  - set **`$psenable = 1`**
  - set **`$gnubin = "gnuplot_binary"`**

  Disable PostScript: in file in `ffcout.cgi` and `ffcoutl.cgi`[1],

  - set **`$psenable = 0`**

- *General*: in file in `ffcout.cgi` and `ffcoutl.cgi`[1],
  - set **`$calcbin = "calculation_binary"`**
  - set WWW home directory
    `$wwwroot = "http://Server_address.WWW_home"`
  - modify the links in the HTML pages in directory `man` to
    `"http://Server_address.WWW_home/man/file.htm"`

  set rights such that user `nobody` may

  - read, write in and execute *`WWW_home_directory`*
  - read and execute the files in *`WWW_home_directory`*

[1] only, if the long double version of the calculation program is available

Manual for the Fluid Flow Calculator

Important!

Stepping through the program

Problems

Miscellaneous

Usage

Please fill in the "Fluid Flow Calculator" form and adjust the values to your own needs. Before choosing numeric values, please fix your data and time units. Please observe that by using merely default values, the calculation program will terminate on error.

- The **Reset** button clears the user input, i.e. shows the default values.
- The **Start** button sends the data to the calculation program and starts it. Depending on the size of the system, this calculation program may take a while to execute. Thank you for your patience!

Upon reception of errors or warnings, or if results seem to be strange, please consult the problem pages. By using the back button of your browser, you can get back to your input and modify it without having to fill in the whole form again! If you however prefer that, please use the Reset button.

The files that are produced may be downloaded by clicking on the corresponding links. They stay on the server until

- they are overwritten by files with same names,
- they are removed by the administrator of the web/computeserver.

back

Units

Due to the broad range of applications for the fluid flow model, it would not be sensible to predefine data and time units. Therefore, this specification has been left to the user completely. The following table shows the ways in which parameters are affected by the chosen data and time units.

Parameter	Unit
Buffer size *K*, thresholds	Data unit
Data rates	Data units/time units
Transition rates	1/time units

However, for VBR 2-state sources and servers, there are some principle relationships between some of the parameters that deserve special attention:

- The *mean burst duration* is given by the corresponding reciprocal transition rate.
- The *mean cycle time* is the total mean burst duration for both states:

$$t = 1/l + 1/m$$

- The *mean burst size* (= mean amount of information within a burst) is given by the product of the mean burst duration and the corresponding data rate.
- Especially for on-off sources ($l^+ = 0$), the buffer size K might be related to the mean burst size in the on-state by a factor k:

$$K = k\, h^+ / m^+$$

back

Quality of Service-related parameters and thresholds

1. **Quality of service (QoS):** The desired output of the calculation has to be specified. One of the following parameters has to be chosen from a list:
   - Probability of saturation: Probability that the input rate exceed the service rate.
   - Loss probability (= default): Probability that a unit of information gets lost due to buffer overflow. The loss probability can either be related to all sources or to single groups of similar sources.
   - Overflow probability of buffer threshold: Probability that the buffer content exceeds a given threshold.

2. **Fluid flow model:** The choice of the fluid flow model determines the size of the buffer and thus the speed, memory requirements and stability of the calculations. Not each model is suited for each QoS parameter: There is no loss in a system in a system with infinite buffer, and the probability of saturation is independent of the existence of a buffer. The standard cases for each parameter are marked by ☑, while additional possibilities are marked with ✓. Arrows describe how the remaining cases are translated into the standard cases.

Model	Prob. of saturation	Loss probability	Overflow prob.
Infinite buffer	↓	↓	☑
Finite buffer (= default)	↓	☑	✓
Buffer-less	☑	✓	↑

3. **Buffer size:** The buffer size needs to be specified for the finite buffer case. In the buffer-less case, this value will be set to 0. In the infinite buffer case, it will be internally set to 1e+307, which is a symbol for infinity. In the buffered case, a negative buffer size will be set to 0, which is followed by a warning. However, a non-numerical input throws error 3.

The following three parameters enable users to specify a ranges of buffer-related parameters for which the calculations are to be carried out - so-called *thresholds*. Inside the calculation program, buffer-related thresholds are the only parameters whose change does not enforce a complete reconstruction of the eigensystem. Moreover, QoS is often plotted versus buffer (threshold) sizes. Therefore, these parameters seem natural to a loop inside the calculation program.

The thresholds have different meanings for different fluid flow models and QoS parameters:

- Buffer-less model: Only one buffer-related threshold is possible, the buffer size (= 0).
- Model with finite buffer:
  - Loss probability: A range of buffer sizes might be given, for which loss probabilities are computed. The range is upper-bounded by the buffer size specification in 3.
  - Overflow probability of buffer threshold: Buffer threshold values between 0 and the buffer size specified in 3 might be given, for which the values of the complementary density function will be computed.
- Infinite buffer model:
  - Overflow probability of buffer threshold: Any non-negative buffer threshold might be given, for which the values of the complementary density function will be computed.

The range of thresholds is specified by

4. **lower threshold** = minimal value (default = 0) - this value is lower-bounded by 0 and upper-bounded by the upper threshold;
5. **upper threshold** = maximal value (default = 0) - this value is lower-bounded by the lower threshold and upper-bounded by the buffer size;
6. **threshold step** size between successive thresholds (default = 1) - this value has to be positive to avoid endless loops.

Both lower and upper thresholds are modified automatically if necessary, followed by a warning. Non-numerical inputs lead to errors 4, 5 or 6, while a non-positive step size throws error 7.

back

Sources

7. A *group of sources* contains sources with similar characteristics and is defined by choosing a **type** (CBR or VBR 2-state) from the corresponding type list. Groups of sources have to be defined one after another without gaps between them. A group *i* (**ID** in the table) whose type is marked with -- is treated as not being defined; this is the default for each group. Valid definitions are:

   - Group 1 alone
   - Group 1 and 2
   - Group 1, 2 and 3

   From these definitions, the number n_g of groups of sources is calculated.

   Invalid definitions are:

   - Group 1 and 3
   - Group 2 alone
   - Group 2 and 3
   - Group 3 alone

   Definitions that include gaps throw an error. This error is unnumbered because this problem only appears in connection to the GUI.

7. If a Quality of Service (**QoS)** has been chosen that might be related to a group of sources, e.g., the loss probability, then the group for which the QoS should be determined might be selected via the corresponding radio button. For the probability of saturation and the overflow probability of a certain buffer threshold, such a setting has no meaning. The default setting is an average QoS for all source types. The selection of a group that has not been defined before (see 7) throws error 12.
8. The **number of sources** n_i^+ within each defined group has to be set. The minimal value is 0, which is assumed by default. A non-numerical value throws error 14, while a negative value throws error 15.
9. For VBR 2-state sources, the **input rate for the "source low" state** l_i^+ has to be given; the default is 0. If the entered value is non-numerical or negative, errors 16 or 17 are thrown. In the case of CBR sources, an entered value will be ignored.
10. The **input rate for the "source high" state** h_i^+ is needed for both source types. This value has to be larger than 0 for CBR sources and larger than the "source low" data rate for VBR 2-state sources; otherwise, error 19 is issued, while a non-numerical input leads to error 18.
11. The **transition rate from the "source low" to the "source high" state** l_i^+ has to be specified for VBR 2-state sources; for CBR sources, this value is ignored. The default value of 0 is not suitable for VBR 2-state sources. If the value is non-numerical or non-positive, errors 20 or 21 are issued.

12. The **transition rate from the "source high" to the "source low" state** m_i^+ is treated in the same way as described under 12; the error codes for on-numerical or non-positive values are 22 or 23, respectively.

back

Servers

14. A *group of servers* contains servers with similar characteristics and is defined by choosing a **server type** (CBR = default or VBR 2-state) from the corresponding type list.

15. The **number of servers** n^- has to be set. The minimal value is 1, which is assumed by default. A non-numerical value throws error 25, while a value smaller than 1 leads to error 26.

16. For VBR 2-state servers, the **server rate for the "server low" state** l^- has to be given; the default is 0. If the entered value is non-numerical or negative, errors 27 or 28 are thrown. In the case of CBR servers, an entered value will be ignored.

17. The **server rate for the "server high" state** h^- is needed for both server types. This value has to be larger than 0 for CBR servers and larger than the "server low" data rate for VBR 2-state servers; otherwise, error 30 is issued, while a non-numerical input leads to error 29.

18. The **transition rate between "server low" and "server high" state** l^- has to be specified for VBR 2-state servers; for CBR servers, this value is ignored. The default value of 0 is not suitable for VBR 2-state servers. If the value is non-numerical or non-positive, errors 31 or 32 are issued.

19. The **transition rate between "server high" and "server low" state** m^- is treated in the same way as described under 18; the error codes for non-numerical or non-positive values are 33 or 34, respectively.

back

Output file

The output of the calculation program is written to a file. The format of this file is such that

- it might be imported into MS Excel for further processing, e.g. for producing graphical output;
- it might be used to produce PostScript files with gnuplot.

The columns of this file are described in the section "Output files".

20. The **name of the output file** has to be given. To simplify the handling of output files on Windows computers, the length of the name has been limited to 8 characters. A default name ffcout is provided. However, this file name should be used with caution if more than one user uses the program at the same time. An extension .xls is attached automatically. This has the advantage that on Windows computers MS Excel can be started within the browser window just by clicking on the file name after the file has been produced.

21. One of the following radio buttons is used to specify **the kind of output** that is desired.
    a. **New file + PostScript.** First, a new output file is written, including the header line that contains abbreviations for the different columns. *A file that has been existing under the same name before is overwritten without check and warning!* This makes it easy to replace erroneous calculations, but might be quite dangerous if different users use the same file name.

       The required steps to import the data into Excel are described under the corresponding keyword. **The file is also displayed so that the user may decide whether (s)he wants to download it**.

       After the Excel file has been produced and if are at least two data lines in the corresponding file, a gnuplot control file under the same name, but with the different extension .gnu will be written by the CGI script. By using this file, gnuplot will produce a PostScript picture under the same name with the extension .ps. In that picture, the x-axis will show the threshold value, while the y-axis represents the QoS parameter under study. The user will be able to download both files. By corresponding settings in the operating system or in the browser, a PostScript viewer (e.g. ghostview) might be started automatically.

    b. **New file.** The same as a), but without the PostScript picture.
    c. **Append to file.** One or more new rows are attached to a file, even if this file has been empty before. *There is no check whether the file already exists!* The new rows are attached without a header line in the beginning. It is up to the user to make sure that the number of groups of sources is not changed while (s)he appends to a certain file, because otherwise, the number of columns will change and thus, some columns will change their meaning. **The file is also displayed.**

The latter option c. is not offered in a PostScript variant, because most probably, it will be used to

study the influence of other parameters than a buffer threshold. However, a gnuplot control file that has downloaded might easily be modified to account for the parameters of interest.

back

Output analysis with Excel (and gnuplot)

Downloading of the Excel-compatible data file on a Windows-PC with Explorer on it causes Excel to start automatically within the browser window, and the data file is displayed.

Important! If a data file is reproduced on the server (by choosing the same file name), Excel might not be willing to show the new contents. If this happens, a new file name should be chosen!

- To assign columns:
  - Mark at least the first column of the table
  - Choose Data → Text to columns
  - Click on Finish
- To fix column widths automatically:
  - Mark the whole table
  - Choose Format → Column → AutoFit Selection
- To produce a diagram (Quality of service versus threshold):
  - Mark the last two data columns
  - Choose Insert → Chart
  - The most suitable chart format is XY (Scatter)

The following table shows the layout of the Excel table for one, two and three groups of sources, both with Excel-specific character and numerical column notions (as needed for gnuplot).

# col., 1 grp.	# col., 2 grp.	# col., 3 grp.	Text in file header	Description
A = 1	A = 1	A = 1	#QoS-par.	QoS parameter (symbol # needed for gnuplot)
B = 2	B = 2	B = 2	Group	Group for which QoS is valid (0 = average)
C = 3	C = 3	C = 3	n1+	Number of sources in group 1
D = 4	D = 4	D = 4	l1+	"Source low" input rate for group 1
E = 5	E = 5	E = 5	h1+	"Source high" input rate for group 1
F = 6	F = 6	F = 6	la1+	"Source low to high" transition rate for group 1 (0 for CBR sources)
G = 7	G = 7	G = 7	mu1+	"Source high to low" transition rate for group 1 (0 for CBR sources)
	H = 8	H = 8	n2+	Number of sources in group 2
	I = 9	I = 9	l2+	"Source low" input rate for group 2

	J = 10	J = 10	h2+	"Source high" input rate for group 2
	K = 11	K = 11	la2+	"Source low to high" transition rate for group 2 (0 for CBR sources)
	L = 12	L = 12	mu2+	"Source high to low" transition rate for group 2 (0 for CBR sources)
		M = 13	n3+	Number of sources in group 3
		N = 14	l3+	"Source low" input rate for group 3
		O = 15	h3+	"Source high" input rate for group 3
		P = 16	la3+	"Source low to high" transition rate for group 3 (0 for CBR sources)
		Q = 17	mu3+	"Source high to low" transition rate for group 3 (0 for CBR sources)
H = 8	M = 13	R = 18	n-	Number of servers
I = 9	N = 14	S = 19	l-	"Server low" service rate
J = 10	O = 15	T = 20	h-	"Server high" service rate
K = 11	P = 16	U = 21	la-	"Server low to high" transition rate (0 for CBR sources)
L = 12	Q = 17	V = 22	mu-	"Server high to low" transition rate (0 for CBR sources)
M = 13	R = 18	W = 23	bufsize	Buffer size
N = 14	S = 19	X = 24	thresh	Threshold size (= buffer size for loss prob./buffer-less model)
O = 15	T = 20	Y = 25	QoS	QoS

For help on how to edit the gnuplot control file, please use gnuplot's own help facilities.

back

Error codes

All of the numbered errors listed below are discovered by the calculation program ff_cmdln.exe. Besides that, there are some errors also reported by the CGI script that calls ff_cmdln.exe, but often with a shorter error message (the texts in brackets are omitted). The parameter numbers in the second-last column refer to the position of the parameter in the command-line call of the calculation program. The positions that the corresponding parameters have depend on the number of source types.

Code	Error	Description	CGI?	Pos.	To do
1	Unknown QoS	An invalid Quality of Service (QoS) specification has occurred		1	Choose either `satu`, `loss` or `over`
2	Unknown model	An invalid fluid flow model specification has occurred		2	Choose either `inf`, `fin` or `bls`
3	(Buffer size) not a number	A non-numeric buffer size specification has occurred		3	Choose a non-negative number
4	(Lower threshold) not a number	A non-numeric lower threshold specification has occurred		4	Choose a non-negative number
5	(Upper threshold) not a number	A non-numeric upper threshold specification has occurred		5	Choose a non-negative number
6	(Threshold step) not a number	A non-numeric threshold step specification has occurred		6	Choose a non-negative number
7	Positive step required	A non-positive threshold step has occurred	✓	6	Choose a real-valued number
8	(Number of groups of sources) not an integer	A non-numeric specification of the number of groups of sources has occurred (negative, character, …)		7	Choose a positive integer
9	No groups of sources defined	A non-positive number of groups of sources has occurred	✓	7	Choose a positive integer; specify groups of sources in ascending order without gaps
10	Too many groups of sources present	A number of groups of sources has occurred that is larger than the internal limit in ff_cmdln.exe		7	Choose a positive integer that is less or equal than `FF_MAX_NO_TYPES` in `ff_const.h`

11	(Group for QoS) not an integer	A non-numeric specification of the type for which the QoS "loss probability" should be determined has occurred		8	Choose a non-negative integer
12	Group for QoS does not exist	The group of sources whose QoS should be calculated does not exist		8	Choose a non-negative integer that is less or equal than parameter 7; do not select QoS for unspecified source types
13	Unknown source type	An invalid source type characterization has occurred		9/15/ 21…	Choose either `cbr` or `vbr2`
14	(Number of sources) not an integer	A non-numeric specification of the number of sources has occurred		10/16/ 22…	Choose a positive integer
15	Positive number of sources required	A negative number of sources has been given		10/16/ 22…	Choose a positive integer
16	(Input rate) not a number	A non-numeric specification of the "source low"-state input rate has occurred		11/17/ 23…	Choose a non-negative number
17	Non-negative input rate required	A negative input rate for the "source low"-state has occurred	✓	11/17/ 23…	Choose a non-negative number
18	(Input rate) not a number	A non-numeric specification of the "source high"-state input rate has occurred		12/18/ 24…	Choose a non-negative number
19	Input rate larger than … required	A data rate for the "source high"-state has been specified that is not larger than the input rate for the "source low"- state	✓	12/18/ 24…	Choose a data rate that is higher than parameter 11/ 17/24…, or choose CBR property
20	(Transition rate) not a number	A non-numeric specification of the transition rate from "source low" to "source high" has occurred		13/19/ 25…	Choose a positive number
21	Positive transition rate required	A non-positive transition rate from "source low" to "source high" has occurred	✓	13/19/ 25…	Choose a positive number

22	(Transition rate) not a number	A non-numeric specification of the transition rate from "source high" to "source low" has occurred		14/20/26…	Choose a positive number
23	Positive transition rate required	A non-positive transition rate from "source low" to "source high" has occurred	✓	14/20/26…	Choose a positive number
24	Unknown server type	An invalid server type characterization has occurred		15/21/27…	Choose either `cbr` or `vbr2`
25	(Number of servers) not a number	A non-numeric specification of the number of servers has occurred		16/22/28…	Choose a positive integer
26	Positive number of servers required	A negative number of servers has been given		16/22/28…	Choose a positive integer
27	(Server rate) not a number	A non-numeric specification of the "server low"-state data rate has occurred		17/23/29…	Choose a non-negative number
28	Non-negative server rate required	A negative data rate for the "server low"-state has occurred	✓	17/23/29…	Choose a non-negative number
29	(Server rate) not a number	A non-numeric specification of the "server high"-state data rate has occurred		18/24/30…	Choose a non-negative number
30	Server rate larger than … required	A data rate for the "high" state has been specified that is not larger than the data rate for the "server low"-state	✓	18/24/30…	Choose a data rate that is higher than parameter 17/23/30…, or choose the CBR property
31	(Transition rate) not a number	A non-numeric specification of the transition rate from "server low" to "server high" has occurred		19/25/31…	Choose a positive transition rate
32	Positive transition rate required	A non-positive transition rate from "server low" to "server high" has occurred	✓	19/25/31…	Choose a positive transition rate

33	(Transition rate) not a number	A non-numeric specification of the transition rate from "server high" to "server low" has occurred		20/26/32 …	Choose a positive transition rate
34	Positive transition rate required	A non-positive transition rate from "server low" to "server high" has occurred	✓	20/26/ 32…	Choose a positive transition rate
35	Type of output not available	A kind of output has been chosen that is not supported		21/27/33 …	Choose `xls` for an Excel table or `xla` for appending to an Excel table
36	Error in the number of parameters	Depending on the number of source types, the number of parameters is given by 22/28/34…			Check whether the set of parameters is complete
> 99	Load too high error code = load in % (100 - 199; from 200 on no further distinction).	The mean arrival rate is higher than the mean service rate, which is critical for the fluid flow models with finite and infinite buffer		7, 9-20/26/32 …	Reduce the load by decreasing • the number of sources • the source data rate(s) • the "source low-to-high" transition rate(s) • the "server high-to-low" transition rate or by increasing • the number of servers • the server data rate(s) • the "server low-to-high" transition rate • the "source high-to-low" transition rate(s)
	Please assign source groups consecutively	There are gaps in the assignment of groups, e.g. group 1 and 3 are defined, but not group 2	✓		Define groups of sources in the order 1, 2, 3 without gaps
	Not enough data for PostScript picture	The thresholds have been chosen (automatically) in a way that there would be only one point in the picture	✓	(2-3) 4-6	Use buffered models and an upper threshold (buffer size) that is at least one threshold step larger than the lower threshold

back

Load

For the buffered fluid flow models, the **offered load** has to be limited below 100 %. This section explains how the load is calculated. The variables are explained in the source and server section, respectively.

For a group i of n_i^+ CBR sources, the mean data rate is given by

$$m_i^+ = n_i^+ h_i^+$$

and for a group of VBR 2-state sources by

$$m_i^+ = n_i^+ (m_i^+ l_i^+ + l_i^+ h_i^+) / (m_i^+ + l_i^+)$$

The mean data rate of all sources becomes

$$m^+ = \Sigma_i m_i^+$$

Similar definitions apply to the n^- servers. Here, the mean service rate for CBR servers is given by

$$m^- = n^- h^-$$

and for VBR 2-state servers by

$$m^- = n^- (m^- l^- + l^- h^-) / (m^- + l^-)$$

The *offered load A* is the ratio of the mean data rate of all sources and the mean service rate:

$$A = m^+ / m^-$$

If this value is larger or equal than one, the buffered fluid flow models cannot be used anymore. In that case, an error code is issued that reflects the rounded load in percent, while loads greater than or equal 200 % are not reported explicitly.

back

Warnings

In some cases, user input has to be corrected or modified. The user will be notified about that by warnings while or after the calculation program is or has been carried out. The warnings are generally unnumbered. Warnings that are also issued by the CGI script are marked accordingly.

Warning	Reason	CGI?	Par.	To do
Model with infinite buffer set	QoS "Overflow probability" (`over`) has been used with the buffer-less model (`bls`)	✓	1, 2	Specify fluid flow model with finite (`fin`) or infinite (`inf`) buffer together with the overflow probability (`over`)
Buffered model set	QoS "Loss probability" (`loss`) has been used with the model with infinite buffer (`inf`)	✓	1, 2	Specify buffer-less model (`bls`) or model with finite buffer (`fin`) together with the loss probability (`loss`)
Buffer-less model set	QoS "Probability of saturation" (`satu`) has been used with the model with finite buffer (`fin`) or infinite buffer (`inf`)	✓	1, 2	It suffices to specify the buffer-less model (`bls`) together with the probability of saturation (`loss`)
Buffer size set to 0	A negative buffer size has been given	✓	3	Specify a non-negative buffer size
Lower threshold set to 0	A negative lower threshold size has been given	✓	4	Specify a non-negative lower threshold
Lower threshold set to buffer size ...	A lower threshold that is larger than the buffer size has occurred	✓	3, 4	Specify a lower threshold that is smaller or equal to the buffer size
Upper threshold set to buffer size ...	An upper threshold that is larger than the buffer size has occurred	✓	3, 5	Specify an upper threshold that is smaller or equal to the buffer size
Upper threshold set to lower threshold ...	An upper threshold that is smaller than the lower threshold has occurred	✓	4, 5	Specify an upper threshold that is not smaller than the lower threshold
Large state space (... states)	A large state space occurred, which may lead to long execution times and numerical instabilities		10/16/22/28 ...	Specify the number of VBR-2 state sources and servers so that the product of the sums of their numbers and one is not larger than `WARN_NO_STATES` in `ff_cmdln.cc` (usually `1000`)

back

Number of states

Time and memory requirements as well as the numerical stability of the calculation program depend heavily on the size of the state space. This section discusses how this size is calculated.

The number of states for n_i^+ CBR sources is given by

- $n_i^+ = 1$

and for n_i^+ VBR 2-state sources by

- $n_i^+ = n_i^+ + 1$

The total number of source states becomes

- $n^+ = \Pi_i \, n_i^+$

The same applies for n^- CBR servers

- $n^- = 1$

or n^- VBR 2-state servers

- $n^- = n^- + 1$

Thus, the size of the state space becomes

- $n = n^+ \, n^-$

A warning is issued if n exceeds the predefined value `WARN_NO_STATES` in `ff_cmdln.cc`. Based on experience, this value has been set to 1000. However, this value represents a kind of weak limit, especially with regard to numerical stability.

Important note: For one group of VBR 2-state sources or servers, the size of the state space should be

- less than about 200 for the standard calculation program `ff_cmdln`
- less than about 500 for the long double version of the calculation program `fflcmdln`

back

Strange Quality of Service (QoS)

On the first glance, "strange" QoS means

- negative probabilities;
- probabilities larger than one;
- values that always stay zero.

Such values may appear in the buffered fluid flow models especially if

- the number of sources and servers become large, which leads to a large state space;
- the number of sources of one group (or servers) reaches the magnitude of 200 (or 500 for the long double variant)
- a large finite buffer size occurs;
- the lowest service rate is still larger or equal to the highest data rate from the sources. In the case of one or more CBR servers, the latter is called *peak rate allocation*.

To find out about the (weak) borders for numerical stability, please

- reduce the number of sources and/or servers;
- reduce the buffer size;
- reduce the server rate(s) so that the total input rate has the chance to exceed the server rate.

Sometimes, a "strange" QoS is not obvious. Therefore, it is recommended to countercheck critical results with fluid flow simulations or numerical integrations of the differential equations. Typical hints that a system gets into numerical trouble are:

- Small variation of input parameters lead to large QoS variations.
- Trends are not unique. If all other parameters remain constant and one parameter is raised, then the QoS should either sink or rise, but not alternate!

back

Software components

The names given in [brackets] are used to invoke the numerically more stable, but slower *long double version of the calculation program* that may not be available on each server.

HTML form `ffcalc.htm [ffcalcl.htm]`

This file is called in a browser and delivers the Graphical User Interface (GUI) for the calculation program `ff_cmdln [fflcmdln]`. It consists of a FORM that is filled in by the user and either reset or sent away via standard input (POST method) to the CGI script `ffcout.cgi [ffcoutl.cgi]`.

The element that are used in this form are the following:

- Select: The user may select an item from a list
- Input of type text: The user may type in text or numbers
- Input of type radio: The user may choose between different possibilities

All the elements are preset with default values.

CGI script ffcout.cgi

This file represents the interface between the GUI and the calculation program `ff_cmdln [fflcmdln]`. It is programmed in the script language Perl and

- receives the data from the GUI, i.e. the HTML form ffcalc.html,
- corrects minor inconsistencies,
- determines the number of source groups,
- checks for obvious input errors,
- fixes the set of input parameters and starts the calculation program,
- starts the calculation program and waits for its execution,
- checks for errors reported by the calculation program,
- and, if no error has occurred, displays the link to the Excel file that has been produced by the calculation program.

If the user has chosen the respective option in the HTML form and if there have at least two QoS values been calculated, the script

- produces control code for gnuplot,

- starts gnuplot to produce a PostScript file
- and issues links to the gnuplot control file and to the PostScript file.

The output of this script in invisible as long as the calculation program is being carried out.

The names of the variables are adapted to those being used within the calculation program `ff_cmdln [fflcmdln]`.

C++ calculation program ff_cmdln [fflcmdln]

This very central program receives its parameters via arguments on the command line, either from the GUI via the CGI script or directly by the user when calling the program from a shell provided by the operating system. This approach has an important advantage: The same program might be used for different kinds of scenarios, i.e. for interactive use via a GUI as well as for batch processes.

The long double version of the program **`fflcmdln`** is numerically more stable, especially for large numbers of sources or servers within one group (200...500). As the data type `long double` needs a special library to be built from, this version may not be built on all computers. Furthermore, it is much slower than `ff_cmdln`.

The total number of parameters and their role depend on the number of groups of sources. They are listed on a special page.

On Windows PCs, the name of the executables should be trailed by **`.exe`**.

back

Parameters for the command-line calculation program

The following table gives an overview over the parameters that `ff_cmdln` [or its long double version `fflcmdln`] has to be called with, together with a specification of values that are supported. The internal variable name in the Perl script is also given. The number of the respective parameter varies according to the chosen number of groups of sources. User-defined values of parameters that are set automatically in certain cases are mostly ignored.

# par., 1 grp.	# par., 2 grp.	# par., 3 grp.	Internal name	Role	Valid entries
1	1	1	`$qostype`	Quality of Service parameter 1. Prob. of saturation 2. Loss probability 3. Overflow prob. of buffer threshold	1. `satu, s` 2. `loss, l` 3. `over, o`
2	2	2	`$ffmodel`	Fluid flow model 1. with infinite buffer 2. with finite buffer 3. buffer-less	1. `inf, I` 2. `fin, f` 3. `bls, b, 0`
3	3	3	`$bufsize`	Buffer size (automatically set to `1e307` in the infinite buffer case, which stands as a symbol for infinity)	≥ `0.0`
4	4	4	`$threshlow`	Lower threshold size	≥ `0.0`
5	5	5	`$threshupp`	Upper threshold size	≥ `0.0`
6	6	6	`$threshstep`	Threshold step size	> `0.0`
7	7	7	`$nogroups`	Number of groups of sources	> `0`
8	8	8	`$losstype`	QoS (loss) that might be observed 1. individually for a certain group 2. for all groups (auto-set to 0 for	1. > `0` 2. `0`

				saturation/overflow)	
9	9	9	`$srctype1`	Source type for group 1 1. CBR (Constant bit rate) 2. VBR 2-state (Variable bit rate)	1. `cbr` 2. `vbr2`
10	10	10	`$n1`	Number of sources of group 1	≥ `0`
11	11	11	`$r01`	"Source low" data rate for group 1 (auto-set to 0 for CBR)	≥ `0`
12	12	12	`$r11`	"Source high" data rate for group 1 1. For CBR: 2. For VBR 2-state:	1. > `0.0` 2. > `$r01`
13	13	13	`$l1`	"Source low to high" transition rate (auto-set to 0 for CBR)	> `0.0`
14	14	14	`$u1`	"Source high to low" transition rate (auto-set to 0 for CBR)	> `0.0`
	15	15	`$srctype2`	Source type for group 2 1. CBR (Constant bit rate) 2. VBR 2-state (Variable bit rate)	1. `cbr` 2. `vbr2`
	16	16	`$n2`	Number of sources of group 2	≥ `0`
	17	17	`$r02`	"Source low" data rate for group 2 (auto-set to 0 for CBR)	≥ `0`
	18	18	`$r12`	"Source high" data rate for group 2 1. For CBR: 2. For VBR 2-state:	1. > `0.0` 2. > `$r02`

	19	19	`$l2`	"Source low to high" transition rate (auto-set to 0 for CBR)	> `0.0`
	20	20	`$u2`	"Source high to low" transition rate (auto-set to 0 for CBR)	> `0.0`
		21	`$srctype3`	Source type for group 3 1. CBR (Constant bit rate) 2. VBR 2-state (Variable bit rate)	1. `cbr` 2. `vbr2`
		22	`$n3`	Number of sources of group 3	≥ `0`
		23	`$r03`	"Source low" data rate for group 3 (auto-set to 0 for CBR)	≥ `0`
		24	`$r13`	"Source high" data rate for group 3 1. For CBR: 2. For VBR 2-state:	1. > `0.0` 2. > `$r03`
		25	`$l3`	"Source low to high" transition rate (auto-set to 0 for CBR)	> `0.0`
		26	`$u3`	"Source high to low" transition rate (auto-set to 0 for CBR)	> `0.0`
15	21	27	`$servtype`	Server(s) type 1. CBR (Constant bit rate) 2. VBR 2-state (Variable bit rate)	1. `cbr` 2. `vbr2`
16	22	28	`$ns`	Number of servers	> `0`
17	23	29	`$r0s`	"Server low" data rate (auto-set to 0 for CBR)	≥ `0`

18	24	30	`$r1s`	"Server high" data rate 1. For CBR: 2. For VBR 2-state:	1. > `0.0` 2. > `$r0s`
19	25	31	`$ls`	"Server low to high" transition rate (auto-set to 0 for CBR)	> `0.0`
20	26	32	`$us`	"Server high to low" transition rate (auto-set to 0 for CBR)	> `0.0`
21	27	33	`$outftype`	Output file type 1. New Excel-compatible file 2. Append to Excel-compatible file	1. `xls` 2. `xla`
22	28	34	`$outfname`	Output file name (automatically truncated to 8 characters; ending `.xls` will be attached automatically)	1…8 characters

back

Web/Compute Server Configuration

The compute server, which carries out the calculation program, should be set up as a web server. This web server should have

- the ability to carry out CGI files;
- a time-out setting that allows for keeping connections open until the calculation program is finished;
- rights for "nobody" to execute programs/write files;
- `Perl` installed;
- a C++ runtime library (not needed, if linked statically);
- the program `gnuplot`, if PostScript pictures are desired.

back

Security and administrative issues

- To hinder third parties to use this software, the web server should be an Intranet server.
- To improve security, put an HTML page at the beginning that calls ffcalc.html and that asks for a user name and a password. Each user could get his account to store his or her files on the web/compute server.
- After having been computed and produced, the files are not deleted from the server automatically. This is a kind of backup, e.g. to enable people to just get a file back if they would have deleted it from their local disk by accident instead of carrying out the corresponding calculation once more, or to be able to append data to a given file. But there are also implicit dangers with this approach:
    - Files that are not needed any more have to be taken away manually or by executing a special script to beware the web/compute server to run out of disk space. Different accounts for different users might ease this task. Each user might be given access to a special script to clean up his/her directory.
    - Two or more people might try to write onto the same file at the same time, especially when the default filename is used. Possibilities to circumvent that problem consist in assigning directories to the users or in simply not providing any default file name at all.

back