A Model for Dynamic Configuration of Light-Weight Protocols

Thomas Plagemann
Martin Vogt
Bernhard Plattner
Thomas Walter

Swiss Federal Institute of Technology Zurich
Laboratory of Computer Engineering and Networks

Abstract

This paper discusses the configuration of light-weight protocols. As opposed to other proposals which introduce a predetermined protocol hierarchy, the configuration is dynamic in the sense that an application can specify requirements of the underlying services and the configuration of a protocol is done with respect to these requirements. The approach discussed in this paper introduces a model for the configuration of protocols comprising of three layers: application, end-to-end communications, and transport infrastructure layer. The dynamic configuration takes place in the end-to-end communications layer. The emphasis of the paper is on the description of the model, a notation for the description of application requirements, and the configuration of protocol entities. The approach is particularly well-suited to make use of any progress in implementations of protocol functions.

1. Introduction

Today's and future high speed transmission facilities and networks have drastically changed our thinking about how communications systems should be structured and implemented. Broadband networks using technologies based on fast packet switching, optical LAN's, Metropolitan Area Networks or fast circuit switching, will offer a flexible *transport infrastructure* with a range of services able to economically support the requirements of various applications. This in turn will strengthen the trend towards *distributed applications*, such that a large variety of such applications with different communication needs will emerge. These needs (e.g. minimum throughput, maximum delay, jitter or the level of error control needed) may not all be directly met by the transport infrastructure; instead, an *end-to-end communications* support will be necessary to provide the quality of services required by the applications. This quite naturally leads to a communications system consisting of a hierarchy of three layers (Figure 1). Layer A corresponds to a set of distributed applications which access the services of the underlying end-to-end communications layer C through the AC-interface. The AC-interface will offer functions that permit the application to specify which type of services are needed and quantify the expected quality of service.

Layer C rests on the transport infrastructure (layer T). It is the task of layer C to map the communication services required by the application to the facilities provided by layer T at the CT-interface. This mapping process encompasses the construction of a suitable protocol entity from a set of protocol functions (step 1) and subsequently the execution of the corresponding protocol machine (step 2).

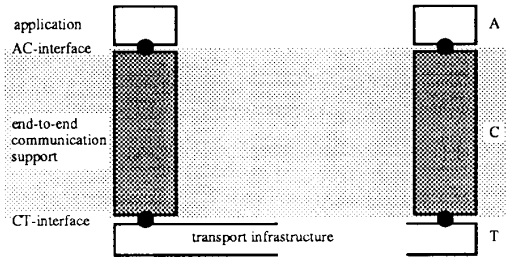

Figure 1 Three layer model

Various authors have proposed similar approaches. Haas [1] describes an architecture consisting of the same three layers and a horizontally oriented protocol for high speed communication (HOPS) built from simple protocol functions. Clark and Tennenhouse [2] investigate "principles to structure a new generation of protocols" and introduce "Integrated Layer Processing" as an protocol engineering principle that allows the implementor to perform all data manipulations in one step (or in several parallel steps) instead of performing them sequentially. O'Malley and Peterson [3] address the issue of constructing a protocol entity from a set of micro-protocols (the equivalent of protocol functions) using the notion of a virtual protocol. Finally, selecting one of several protocols according to the requirements of the application and the

0-8186-2755-7/92 $03.00

service provided by the underlying network layer is used in the ISO transport protocol standard [4].

In this paper we consider step 1 of the two mapping steps mentioned above. The construction of a suitable protocol entity is called its *configuration* which is *dynamic* in the sense that a configuration is created for a given set of application requirements and a specific range services offered by the transport infrastructure. Assuming that the configuration process produces a protocol entity which is optimally adapted to what is needed, this protocol entity will represent a *light-weight protocol* [5], as all unnecessary protocol functions will not be present[1].

The next section will discuss a refined model on which the configuration step is based, while section 3 will show our approach to formally specify the application requirements and the services of the transport infrastructure. Section 4 outlines the configuration step and is followed by an example that should demonstrate the feasibility of our approach.

2. A model for the configuration of light-weight protocols

Our model splits communication systems into the three layers A, C and T (Figure 1). End systems communicate with another via layer T, the transport infrastructure. The transport infrastructure (which is not the same as the OSI transport layer) represents the existing and connected communications infrastructure. The services of the transport infrastructure (layer T services) are available at the CT-interface. Tschudin [6] calls layer T services "anchored instances". Such services may be e.g. the services of the DQDB layer in an IEEE 802.6 MAN or the DoD Internet Protocol. In layer C the end-to-end communication support adds to the layer T services in such manner that at the AC-interface a full set of services is provided to run distributed applications (layer A).

Distributed applications invoke the communications system by naming a layer C service and specifying the required properties of the service (including its quality) and the communication partner(s). We provide a formal language **L** for the specification of application requirements. L enables us to deal with the wide range of application requirements, mainly introduced by new multi-media applications. Furthermore L allows the application to specify the relative importance of its different requirements and to formulate contradictory requirements. Section 3 describes the language L and its use in our model. The application invokes the communication system with a triple <SC, e_{AR}, cp> where SC defines the layer C service, e_{AR}, an expression in the language L, specifies the application requirements and **cp** identifies the communication partner(s).

Layer C protocols are decomposed into protocol functions according to their functionality. Protocol functions are defined by their semantics and their interfaces. They can be implemented in multiple ways, by different algorithms, as software or hardware solutions. We call the implementation of a protocol function a *module*. Modules implementing the same protocol function are characterized by different properties. This may be different throughput figures or different degrees of error correction or detection. For instance the protocol function 'error control' could be implemented by usage of a single parity bit or by the CRC-CCITT algorithm. While the computation of the parity bit is very simple and fast, the probability of error detection in a block is only 0.5. In contrast the CRC-CCITT is computationally costly (i.e. exhibits a reduced throughput), but has much better error detection capabilities. The properties of module M are specified by an expression e_M in the language L. Each protocol function PF is related to a finite set of modules and module descriptions $\{(M_1, e_{M1}), .., (M_k, e_{Mk})\}$.

It should be noted that layer T services may be seen as a specific type of protocol function. Like protocol functions, layer T services are described by their semantics and interfaces. The quality of a layer T service ST, expressed by e_{ST} in L, depends on the current communication partner(s) cp. Like multiple modules per protocol function and their different descriptions, one layer T service ST may be characterized by multiple different expressions $\{(cp_1, e_{ST1}), .., (cp_i, e_{STi})\}$. To illustrate this, consider the service of the FDDI MAC layer as the layer T service. If both end systems are stations of the same FDDI ring the properties of the layer T service will differ from the case when both end systems are stations in different FDDI rings connected by a slow point-to-point link. Higher error probabilities, decreasing throughput and increasing delays (in relation to the first case) will be noted in the second case. It is assumed that the layer T service is able to determine its properties depending on the communication partner(s) cp.

Each layer C service SC is composed of a set of protocol functions on top of a layer T service ST. Dependencies like data dependencies define an (partial) order on these protocol functions. If multiple layer T services $ST_1,..,ST_i$ may be used, there is one functional decomposition of SC for each ST, denoted by func_decomp(SC) = $\{\{PF_{11}, PF_{12}, ..., ST_1\}, ..,\{PF_{i1}, PF_{i2}, ..., ST_i\}\}$. To create a protocol entity each protocol function must be instantiated by one of its implementations (modules). The order of the protocol functions, module constraints, and global con-

[1] We assume that individual protocol functions are designed and implemented according to the state of the art in high speed protocols.

straints are defining the structure of the protocol entity, i.e. a protocol graph. A module constraint for instance may be the fact that a module is an external board and may be executed in parallel to the CPU. An important global constraint is the maximum degree of parallel computations in the end system. Figure 2 illustrates an example of a protocol graph with a degree of 3 in parallelism.

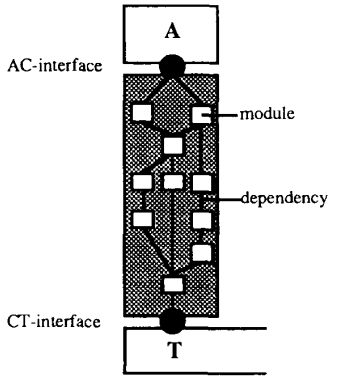

Figure 2 Example of a protocol graph

The task of the configuration process is to select suitable modules such that the resulting protocol entity fulfills the application requirements. The properties of the configured protocol entity can be directly computed from the properties of the layer T service e_{ST} and the properties of the selected modules $\{e_{M1}, .. , e_{Mk}\}$ according to the dependencies emphasized by the protocol graph. The properties of the protocol entity **ep** are expressed in the language L, like e_{ST} and the module descriptions e_{Mi}.

To ascertain that a protocol entity fulfills the application requirements the two expressions e_{AR} and ep must be compared. An exact match of e_{AR} and ep cannot be expected. If a protocol entity fulfills the application requirements we say the protocol entity is in *compliance* with the application requirements. Section 3.3 describes the calculation of the *compliance degree* **cd**. The protocol entity with the highest compliance degree cd of ep and e_{AR} will be the best configurable protocol entity according to the application requirements.

3. Formal specification of application requirements

The previously described model deals with the dynamic configuration of protocols tailored to requirements specified by applications (or users). It is the purpose of this section to introduce a formal language and framework for the unambiguous description of application requirements and modules to assist the configuration process.

For the specification of protocols several formal description techniques (FDT) have been proposed, e.g. SDL [7] and LOTOS [8]. These FDT's allow the specification of behavioral properties of a protocol only, and thus are insufficient for the usage in the context of dynamic configuration of protocols. It seems to be essential to invent a formal method well-suited to be used in our framework.

The language **L** is defined for the description of application requirements, characterizations of modules, transport infrastructures and protocol entities. The basic notation is introduced in Section 3.1. Emphasis is put on the application of L. Particularly important is the computation of characterizations of protocol entities and the definition of compliance which are discussed in Section 3.2 and 3.3.

3.1 The language L

The main idea is that properties of layer T services, modules, protocol entities, and application requirements can be described as combinations of a type, which uniquely identifies a property provided by a protocol entity, module, or the transport infrastructure or recommended by an application, and a value, which gives an indication on the degree this property is satisfied by a protocol entity, module, or transport infrastructure or required by the application.

Our definition of L starts out from a pre-defined set of *attribute types* (e.g. throughput, jitter, cost, etc.) referred to by $A = \{a_1, ..., a_n\}$, a set of *relation symbols* $\{\leq, \geq, =\}$, and for each attribute type a_i a set of *values* V_i with a particular element $\perp$. The value sets are partially ordered with a least element $\perp$. Furthermore, we assume for every attribute type the existence of a set of *functions* $F_{a1}, ..., F_{an}$ defined over the set of attribute values. For every attribute type a particular function, called *weight*, is defined over the value sets as domains and the real numbers as range. The set of all these weights is referred to as $W = \{w_{a1}, .., w_{an}\}$.

Application requirements: An application may specify application requirements by a tuple $e_{AR} = <ar_1, ..., ar_n>$ which is referred to as an expression over L. Each ar is defined as ar = <at, [≤ | ≥ |=] av | *, w> consisting of an attribute type at ∈ A, optionally a relation symbol followed by a value av ∈ V_{at} or a * as "don't care", and a weight w ∈ W. If a value is explicitly specified, instead of the *, then this constraint has to be met by a protocol entity. A relation symbol specifies whether this constraint should be met at least (≤, ≥) or exactly (=). Note that whether ≤ or ≥ is to be used is dependent on the interpretation of the attribute, e.g. throughput (≥) or error rate (≤). The weight defines the relative importance of the requirement, particularly with respect to other application requirements.

Intuitively, an application requirement defines the quality of service needed from the applications or users point of view. Not all application requirements must equally be satisfied. In L we have two possibilities to make this explicit. First, by assigning a relation and value with an attribute type. This enables the application or user to introduce so-called *knock-out conditions* in the sense, that this requirements must be fulfilled at least to the degree indicated. Second, by specifying a weight. The more important the attribute is the higher is the value returned. Generally, a fine tuning to the requirements of the application is possible by specifying appropriate weights.

Whether or to what extent requirements can be matched depends on the available modules and the transport infrastructure. To enable the protocol entity configuration process it is necessary to provide a characterization of layer T services and modules compatible with the characterization of application requirements.

Characterization of layer T services and modules: A feasible approach is to extract from the definition of application requirements parts which are suitable to describe layer T service characteristics and characterizations of modules.

In terms of L a particular layer T service characteristic is defined by a tuple $t = <at, av>$ with $at \in A$ an identification of a attribute type and $av \in V_{at}$ the characteristic value for at. The characterization of all layer T services is given by a combination of tuples t for all attribute types in A: $e_{ST} = <t_1, ..., t_n>$.

For an attribute type not applicable to the characterization of layer T services the associated value is the least element $\perp$. This overhead in the characterization of layer T makes the computation of characterization of protocol entities more convenient (Section 3.2).

After the previous paragraphs the characterization of modules M_i is straightforward and is the combination of tuples $m = <at, f \mid [\leq \mid \geq \mid =] av>$ with $at \in A$, $f \in F_{at}$, and $av \in V_{at}$ to form $e_{Mi} = <m_1, ..., m_k>$

The differences to the definition of e_{ST} are that we do not assume to specify a value for all attribute types and that we also allow functions as attribute values. This provides a certain flexibility in the description. The characterization of a module may be made independent of the context, determined by the hierarchy of modules which define a protocol entity bound to one out of several layer T services. (E.g. the reduction in throughput caused by a module is a constant factor.) During the configuration process this function can be evaluated on the attribute value of anyone of the modules. This is described on a more detailed level in the Section 4.

3.2 Protocol entities

As illustrated in Figure 2 a protocol entity is configured on top of layer T and consists of a set of modules. The characteristics of the protocol entity have to be checked against the requirements imposed by the application. Therefore, it becomes necessary to investigate the problem how to derive a characterization of the configured protocol P from the characterization of its components.

We can assume that e_{ST} and e_{Mi} for each module M_i are known. e_P, the characterization of protocol P in terms of L, is determined by: $e_P = pc(e_{ST}, e_{M1}, ..., e_{Mn})$. Function pc is computed as follows: According to the dependency graph of modules M_1, M_2, ... (Figure 2) the characterization of modules M_i are derived from their own characterization e_{Mi} and the characterization of all modules e_{Mk}, $e_{Mk'}$ ... they depend on. If a module is directly connected to the transport infrastructure, e_{ST} is used in the computation. A new e'_{Mi} is computed which substitutes the old e_{Mi}. The tuples of e'_{Mi} are determined by the following rules:

- if e_{Mi} explicitly specifies a value this value is used in e'_{Mi},
- otherwise the specified function is evaluated on the values of all e_{Mk}, $e_{Mk'}$ ....

This procedure is repeated for each attribute and on all modules. Ultimately we arrive at a unique characterization of P given by e_P which comprises an attribute tuple for all attribute types in A.

3.3 Optimal protocol entities

This section is concerned with the definition of the compliance relation *com*, which determines whether a configured protocol entity P complies with the application requirements e_{AR}, and the selection of the optimal protocol entity P_{best} out of a set of feasible protocol entities $\{P_1, ..., P_k\}$.

In a first step it is checked that all attribute values of e_{Pi} comply with the corresponding attribute values in e_{AR}; the following rules are used:

- any value complies with * and
- a value in e_{Pi} has t satisfy the condition specified in e_{AR}.

The set of complying protocol entities which results from this selection, is referred to by $\{P_{i1}, ..., P_{in}\} \subseteq \{P_1, ..., P_k\}$. In a second step the *compliance degree (cd)* is computed for each remaining protocol entity in order to determine the best protocol entity. For all attributes ar of e_{AR} the weight w_i is computed on the value v_j of the corresponding attribute of e_P which results in a real denoted by $r_h = w_i(av_j)$. The compliance degree of protocol entity

$P_i \in \{P_{i1}, ..., P_{in}\}$, denoted by cdp_i, is the sum of all r_l for all application requirements $e_{AR} = <ar_1, ..., ar_n>$:

$$cd_{pi} = \sum_{h=1}^{n} r_h$$

Based on compliance degree the selection of P_{best} is reduced to determine the maximal cdp_i out of the set $\{cdp_{i1}, ..., cdp_{in}\}$: $P_{best} := \max(cdp_{i1}, ..., cdp_{in})$.

4. Protocol configuration

The application call <SC, e_{AR}, cp> at the AC-interface initiates the configuration process. In the first place all functional decompositions of the layer C service SC are needed; func_decomp(sc) = $\{FD_1, ..., FD_k\}$ = $\{\{PF_{11}, PF_{12}, ..., ST_1\}, .., \{PF_{k1}, PF_{k2}, ..., ST_k\}\}$. Subsequently each functional decomposition $FD_i \in \{FD_1, ..., FD_k\}$ will be handled in the same way as described hereafter.

Based on the functional decomposition of SC a corresponding protocol entity P could be constructed by instantiating all protocol functions with one of their modules. The combination of all instantiations produces all possible protocol entity configurations according to the functional decomposition, denoted all_combination(FD) = $\{P_1, .., P_n\}$. The next task is the determination of the properties of the protocol entities $\{P_1, .., P_n\}$. As described in section 3.2, ep_i is a function of the properties of the layer T service e_{ST} and the module descriptions e_{Mi} $ep_i = pc(e_{ST}, e_{M1}, ..e_{Mh})$. While the module descriptions e_{Mi} are known, the properties of the layer T services depend on the communication partner(s) cp and are available at the layer CT-interface; we assume that e_{ST} may be determined with a call to a function get_prop(ST, cp) available at the CT interface. The last step of the configuration is to check that ep_i does not violate any knock-out condition of e_{AR} and - if this is granted - to compute the compliance degree of ep_i with e_{AR}, $cdp_i = comp\_cd(e_{AR}, ep_i)$. The protocol entity P with the highest compliance degree is the result of the configuration process. Figure 3 summarizes the configuration process in pseudo code.

The number of possible protocol entities depends on the number of modules per protocol function |PF| and the number of functional decompositions (i.e. the number of usable layer T services) denoted |SC|. The product of all $|PF_{nj}|$, with $PF_{nj} \in FD_n$, defines the number of configurable protocol functions on top of ST_n. Altogether the configuration process must consider

$$N = \sum_{i=1}^{|sc|} \prod_{j=1}^{|FDi|} |PFj|$$

possibilities. It is obvious that such a number of real time computations cannot satisfy high speed requirements. It is important to realize that a major part of the configuration process can be done in advance.

```
input: <SC, e_AR, cp>

{PG_1,...,PG_k} = get_protocolgraphs(SC);
FOR_ALL PG_i ∈ {PG_1,...,PG_k} DO
    e_ST = get_prop(ST, cp);
    {P_1, .., P_n} = all_combination(PG);
    FOR_ALL P_j ∈ {P_1, .., P_n} DO
          ep_i = pc(e_ST, e_M1, ..e_Mh);
    END FOR_ALL
    FOR_ALL ep_i DO
          IF com(e_AR, ep_i) = TRUE THEN
                cd_i = comp_cd(e_AR, ep_i);
          END IF
    END FOR_ALL
END FOR_ALL
P = max(cdp_i1, ..., cdp_in);

output: P
```

Figure 3 Configuration process

Most information used by the configuration process is known at installation time. Specifically, this pertains to the offered layer C services, the functional decompositions of the services, their protocol functions, modules and module descriptions. The application requirements e_{AR} and the communication partner(s) cp are given at the time of the application call and are not known before. In general e_{ST} depends on the communication partner(s) cp and must be acquired at real time from the CT-interface. Therefore it is important to prepare the configuration process in order to reduce the necessary real time computations to a minimum.

We distinguish the cases of homogeneous and heterogeneous transport infrastructures, networks respectively. In homogeneous networks (one single network) all properties of layer T services e_{ST} are known at installation time because e_{ST} is independent of the communication partner(s) cp. In this case we may prepare a list of all possible protocol entities and their properties for each service, SC_1: $\{(P_1, ep_1), (P_2, ep_2), .., (P_k, ep_k)\}$, SC_2: $\{(P_1, ep_1), (P_2, ep_2), .., (P_h, ep_h)\}$,... etc. After the application call <SC_2, e_{AR}, cp> the configuration process calculates the compliance degree for all $ep_i \in \{ep_1, .., ep_h\}$ with e_{AR} and selects the maximum.

In the heterogeneous case (more than one network type) e_{AR} depends on cp. But we could collect a representative set of possible e_{AR} in advance. After the classification of this set we may prepare a list of all possible protocol entities and their properties per service and e_{ST} class. After the application call <SC_2, e_{AR}, cp> the configuration process determines the current e_{ST} = get_prop(ST, cp), classifies the properties, determines compliance and calculates the compliance degree for all $ep_i \in \{ep_1, .., ep_l\}$ with e_{AR} of the corresponding class and selects the maximum.

5. Implementation approach - sample scenario

Now let us illustrate the configuration process with an example. It shows how a protocol for the transfer of a prerecorded video-sequence (1 GByte uncompressed data) is configured. We assume two machines, both connected to the same DQDB - MAN. The process on the machine requesting the transfer of data from the other machine is described in detail. This example will show clearly, that we can configure a protocol which is better suited for this application than the usual FTP/FTAM, which require error free transmission or real-time video-transfer which needs an isochronous service.

First, we look at the components for the protocol configuration namely the triple <SC, e_{AR}, cp>, the available modules M_i and the layer T services ST. Second the configuration process itself is described. The full protocol configuration scheme is shown, although only a limited set of items is included in the description, specifically, the application requirements are limited to the attribute set A = {transfer mode (**tm**), throughput (**th**), error rate (**er**), cost (**c**)}. The attribute values for each type are defined as follows: V_{tm}={⊥, asynchronous, synchronous, isochronous}, V_{th}={⊥, 0, 1, 2, ...}, V_{er}= {⊥, 1, ..., 0}, V_c={⊥, ..., 1, 0}.

AC-interface: The first item we consider is the AC-interface. The service needed on the machine considered is 'File Transfer'. This transfer will neither require error free data transmission nor isochronous transport, but low cost is desired. The throughput is also not critical. So the application requirements e_{AR} (including the weight functions) for the configuration are <<tm, ≥ asynchronous, $w_{tm}(tm)=0$>, <th, ≥ 10^6 Bits/sec, $w_{th}(th)=\log(th)$>, <er, ≤ 10^{-9}, $w_{er}(er)=-0.01*\log(er)$>, <c, *, $w_c(c)=-c*100$>>. The communications partner cp is always the corresponding application on the other machine.

	mode	throughput [Mbit/s]	error rate	cost/ Gbyte
ST_1	isochronous	≤ 1	≤ 10^{-12}	1
ST_2	isochronous	≤ 10	≤ 10^{-12}	5
ST_3	isochronous	≤ 100	≤ 10^{-12}	10
ST_4	asynchronous	≤ 10	≤ 10^{-12}	1
ST_5	asynchronous	≤ 100	≤ 10^{-12}	5
ST_6	asynchronous	≤ 500	≤ 10^{-12}	10

Table 1 Offered services

CT-interface: The CT-interface results from the connection to the DQDB - MAN. We therefore assume a maximum bandwidth of 622MBit/sec, and a guaranteed error rate of 10^{-12}. The cost of the transfer per GByte are specified by the provider depending on throughput and transfer-mode. Table 1 summarizes our assumptions on e_{ST}.

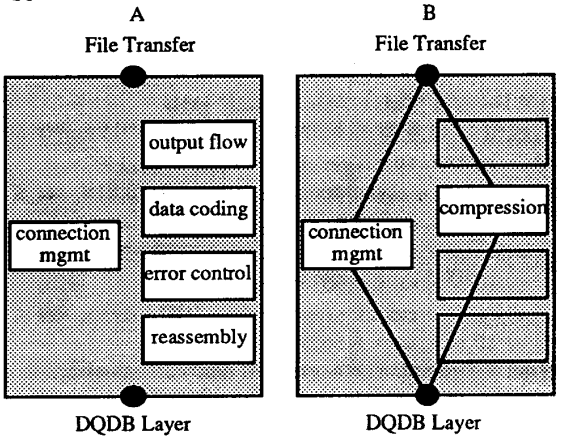

Figure 4 Structure of File Transfer

Protocol functions and modules: We consider only the protocol graph resulting for file-transfer - DQDB layer. The protocol functions and their available modules are described briefly:

output flow: This function assures a maximum jitter at the output of the configured protocol. There is only one module implementing this function. It delivers isochronous output but has disadvantageous influence on delay and throughput. e_{M1}=<tm=isochronous, th*0.9>

data coding: This function does data conversion to the required format. Three modules are available: Text Translation ASCII - EBCDIC, ASN.1 encoding, and compression. Only compression is interesting for our example: we assume that it is able to do compression at a rate of 1:1,5. This increases the throughput but also increases jitter. The exact values are determined at installation time. e_{M2}=<th*1.5, c/1.5>

reassembly: This function does the reassembly of MAC frames segmented for the transfer over DQDB. There is only one module for this function. It decreases the throughput and increases the jitter. e_{M3}=<th*0.9>

error control: This function is responsible for error reduction to the residual error rate allowed. Two modules are available: The first (M_4) calculates a CRC and requests a retransmission if errors are detected while the other (M_5) does forward error control. e_{M4}=<th * 0.9, er *10^{-6}>, e_{M5}=<th*0.9, er*10^{-4}>

Configuration process: In our example the (identical) functional decomposition for all six ST results in four protocol functions. Therefore we get

$$\sum_{i=1}^{6}\prod_{j=1}^{|PDi|}|PFj| = 6\prod_{j=1}^{4}|PFj| = 6 \cdot 2 \cdot 2 \cdot 2 \cdot 3 = 144$$

possible protocol entities. Now all ep_i must be calculated:

$P_1 = \{ST_1, M_1, M_2, M_3, M_4\}$

$ep_1 = ((((e_{ST1} \cdot e_{M3}) \cdot e_{M4}) \cdot e_{M2}) \cdot e_{M1}) =$

$= (((<<tm = asynchronous>, <th \geq 106*0.9>, <er \leq 10^{-12}>, <c = 1>> \cdot e_{M4}) \cdot e_{M2}) \cdot e_{M1}) =$

$= ((<<tm = asynchronous>, <th \geq 106*0.81>, <er \leq 10^{-18}>, <c = 1>> \cdot e_{M2}) \cdot e_{M1}) =$

$= (<<tm = asynchronous>, <th \geq 106*1.215>, <er \leq 10^{-18}>, <c = 0.667>> \cdot e_{M1}) =$

$= <<tm = isochronous>, <th \geq 106*1.093>, <er \leq 10^{-18}>, <c = 0.667>>$

To get the compliance degree, all attribute values are inserted into their respective weight functions:

$cd_1 = w_{tm}(isochronous) + w_{th}(10^6*1.093) + w_{er}(10^{-18}) + w_c(0.667) =$

$= 0+\log(10^6*1.093)-0.001*\log(10^{-18})-100*0.667 =$

$= -60.7$

This calculation is done for all possible protocols. All of them are in compliance with the application requirements. As result, $P_{40} = \{ST_4, M_2\}$ delivers the highest compliance degree, cd_{40}=-59.6, and is therefore selected (Figure 4 B). Only the compression module M_2 is needed in the configured protocol, while the lowest cost throughput class ST_4 is used.

6. Conclusions

In this paper, we introduce a three layer model of communication systems for the dynamic configuration of light-weight protocols. The functional decomposition of layer C services and the formal specification of application requirements, module properties and constraints are the main features of the model. For the specification of application requirements and properties of protocol functions we introduced a new language L with specific capabilities. Driven by expressions in L the configuration process will create a light-weight layer C protocol entity. The non-layered structure of layer C enables us to directly take into account the requirements of a specific application and the properties of the layer T service. Thus, only necessary functionality will be added to the layer T service. The goal of the configuration process is to satisfy the application requirements. We have shown that most of the computations involved in the configuration process can be done in advance without losing the benefits of a dynamic configuration.

The description of the model emphasizes the protocol configuration at the initiator, because the requirements of the initiator must be the decisive factor of the configuration. Therefore the protocol configuration at the responder, which must be done in an analogous way, remains open here. Likewise we do not discuss the influence of intermediate systems like bridges and gateways in the case of a layer T service located lower than layer 3 in the OSI reference model.

It should be stressed at this point that at present no implementation is available, which is one of the next steps in our work. A prototype in a simulation environment will be implemented to study further the configuration process itself. In parallel we will design and implement layer C protocol functions as building blocks for future layer C protocol entities. With this practical experience we intend to identify and improve weak aspects of the presented framework.

References

[1] Haas, Z.: "A Communication Architecture for High-speed Networking", in: Proceedings of IEEE INFOCOMM '90, Ninth Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE Computer Society Press, Los Almos, California; Volume 2, Jun. 1990, pp. 433-441.

[2] Clark, D.D. Tennenhouse, D.L.: "Architectural Considerations for a New Generation of Protocols", in: ACM SIGCOMM Computer Communication Review; Volume 20, Number 4, Sep. 1990, pp. 200-208.

[3] O'Malley, S.W., Peterson, L.L.: "A Highly Layered Architecture for High-Speed Networks", in: Protocols for High-Speed Networks, II, Majory J. Johnston (Editor), Elsvier Science Publishers B.V. (North-Holland), 1991, pp. 141-156.

[4] ISO 8072, CCITT X.214 "Transport Service Definition".

[5] Doeringer, W.A., Dykeman, D., Kaiserswerth, M., Meister, B.W., Rudin, H., Williamson, R.: "A Survey of Light-Weight Transport Protocols for High-Speed Networks", in: IEEE Transactions on Communications; Volume 38, Number 11, Nov. 1990, pp. 2025-2039.

[6] Tschudin, C.: "Flexible Protocol Stacks", in: SIGCOMM '91 Conference, Communications Architectures & Protocols, Zürich, Switzerland, Computer Communications Review; Volume 21, Number 4, Sep. 1991, pp. 197-204.

[7] Z.100, "Z.100: Specification and Description Language SDL, CCITT, 1989.

[8] ISO 8807, "LOTOS - Language for the temporal ordering specification of observational behavior", ISO, 1988.