7

CoRA - A Heuristic for Protocol Configuration and Resource Allocation

Thomas Plagemann, Andreas Gotti and Bernhard Plattner

ETH Zurich, Computer Engineering and Networks Laboratory (TIK)
Gloriastrasse 35, CH-8092 Zurich, Switzerland

Abstract

Da CaPo (**Dyn**amic **C**onfigur**a**tion of **Pro**tocols) provides an environment for the dynamic configuration of protocols. Implementations of single protocol mechanisms (called modules) represent the building blocks. This paper describes the heuristic CoRA (**Co**nfiguration and **R**esource **A**llocation) developed in the context of Da CaPo. CoRA configures protocols at runtime with respect to application requirements, properties of offered network services and available resources in the end systems. The goal of the configuration is to support a wide range of Quality of Service (QoS) requirements with protocols that are optimally adapted to what is needed (i.e., to increase protocol performance by decreasing protocol complexity). Generally, the problem of protocol configuration is quite complex because the set of all possible configurations might be rather large. The classification of building blocks and a measure for the resource usage of building blocks are combined in a structured search approach enabling CoRA to find suitable configurations under real-time constraints.

Keyword Codes: C.2.2, D.2.1, I.2.8
Keywords: Network Protocols, Requirements/Specifications, Problem Solving, Control Methods, and Search

1. Introduction

Recent developments in data communications are dominated by advances in two (mutually spurring) areas: high-speed networking and distributed multimedia applications. It is well known that the communications bottleneck in modern high-speed networks is located in the end system. Multimedia applications increased the set of different requirements (in terms of throughput, end-to-end delay, delay jitter, synchronization, etc.). These needs may not all be directly met by the networks; end system protocols have to enrich network services to provide the quality of service (QoS) required by applications. Obviously, fixed end system protocols are not able to support the wide range of different application requirements on top of current networks (ranging from modem lines up to gigabit networks) without including overhead (i.e., unnecessary functionality) for multiple combinations of the cross-product application requirements and networks.

The aim of the Da CaPo (**Dyn**amic **C**onfigur**a**tion of **Pro**tocols) project is to improve the described situation by configuring end system protocols. Protocols will be optimally adapted to application requirements, to offered network services and to available resources in the end systems. Configuration serves to support a wide range of QoS requirements and to increase protocol performance by decreasing protocol complexity. The properties of network services and end sys-

G. Neufield et al. (eds.), *Protocols for High Speed Networks IV*

tem resources as well as the application requirements are described in a common syntax. We developed a heuristic called CoRA (**Co**nfiguration and **R**esource **A**llocation) to configure suitable protocols at runtime. Mainly, a classification of building blocks and their resource usage are considered in CoRA to reduce the complexity of the configuration task and so decrease configuration time.

Application and QoS driven protocol tailoring and configuration are supported by different systems at different degrees. UNIX System V streams [1] enable applications to configure software modules to protocols at runtime. Haas [2] describes a horizontally oriented protocol for high speed communications (HOPS) built from simple, user selected, protocol functions. O'Malley and Peterson suggest in [3] a complex protocol graph of micro-protocols and virtual protocols. Virtual protocols direct packets through the protocol graph, with each path in the graph corresponding to one protocol configuration. However, none of the previously listed approaches supports automatic mapping of application requirements onto protocol configuration. In extended XTP (XTPX) [4] QoS parameters are mapped onto XTPX procedures and parameters to adapt the protocol machine; while in the Quality of Service Architecture (QoS-A) [5] QoS specifications from service users are classified and mapped onto profiles to tailor some mechanism of the protocol machine to be used. ADAPTIVE [6] performs a similar approach: the configuration process examines the application requirements and attempts to match them to a pre-configured transport service class. The Function-Based Communication SubSystem (F-CSS) [7] offers four pre-defined service classes to applications without special service requirements. More demanding applications can use a variety of service parameters (including QoS parameters) to specify their particular requirements. A configuration process composes specially tailored protocol machines, based on application requirements and information in a protocol resource pool. Such an automatic selection of suitable protocol configurations at runtime is difficult because of its complexity [8]. In this paper, we present an approach which efficiently solves the configuration task in Da CaPo.

A second potential problem which is generic to all approaches that use dynamic protocol configuration (which however is not of relevance for CoRA) is the overhead introduced by the flexible framework. Naturally, general approaches have increased overhead in comparison to monolithic implementation, but within Da CaPo we kept the overhead small by embedding protocols in a specially tailored runtime environment [9].

In this paper, we concentrate on the design and implementation of CoRA. The next section introduces the three layer model, the foundation of Da CaPo and section 3 briefly describes the architecture of Da CaPo. In section 4, we illustrate the basic ideas of CoRA and describe its implementation in section 5. Performance measurements of CoRA are presented in section 6. Section 7 summarizes this work and indicates future work.

2. Three layer model

Da CaPo is based on a three layer model [10] which splits communication systems into the layers A, C and T (Figure 1). End systems communicate with each other via layer T, the transport infrastructure. The transport infrastructure represents the existing and connected communication infrastructures offering end-to-end connectivity. The service of layer T is a generic service and might correspond to layer 2a, 2b, 3 or 4 services in the OSI Reference Model. In layer C the end-to-end communication support adds functionality to the T services such that at the AC-interface services are provided to run distributed applications (layer A).

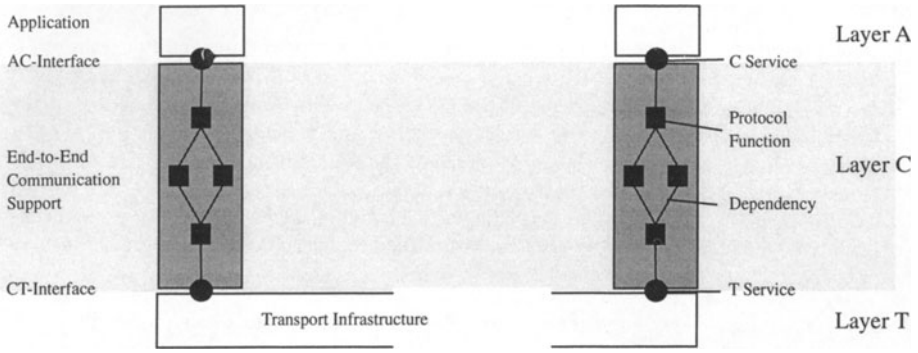

Figure 1. Three layer model

Layer C is decomposed into protocol functions instead of sub-layers. Each protocol function encapsulates a typical protocol task like error control, flow control, encryption and decryption, presentation coding and decoding, etc. Data dependencies between protocol functions, arranged on top of a T service, define a partial order on protocol functions and are specified in a protocol graph. A protocol graph is an abstract protocol specification which has to be defined by a protocol engineer. Independence between protocol functions is directly expressed in the protocol graph and indicates the possible parallel execution of these protocol functions. If multiple T services can be used, there is one protocol graph for each T service to realize a layer C service.

Protocol functions can be accomplished in multiple ways, by different protocol mechanisms, as software or hardware solutions [11]. We call the implementation of a protocol function a *module*. Modules implementing the same protocol functions are characterized by different properties, e.g., different throughput figures or different degrees of error detection and correction. To configure a protocol each protocol function in a protocol graph must be instantiated by one of its modules.

Four types of information are relevant for the configuration process: protocol graphs, available modules, module and T service properties, as well as application requirements. Protocol graphs and information on available modules for the protocol functions are stored in a local database. Furthermore, the database contains specifications of module properties describing the influence of single modules on the offered QoS. Application requirements are specified from applications within connection establishment requests.

We use a common syntax to describe the properties of modules, T services, and application requirements. We call this language *L*. All descriptions in L are based on tuples of attribute types and values or functions. The attribute types are elements of an extensible set of attribute types denoted $\mathcal{A}$. $\mathcal{A}$ contains types like "throughput", "delay", "delay jitter", and "packet loss probability". For each attribute type there is a value set $\mathcal{V}$ with an associated relation. Partially ordered value sets are associated with the relation "$\geq$" or "$\leq$" and unordered value sets with the relation "=". All expressions in L comprise the previously introduced basic elements. In particular, the following four types of expressions are supported:

- *T service properties* (denoted e_{ST}) are simply specified by tuples of attribute types and attribute values: $e_{ST} = <st_1, ..., st_n>$ and st_i= (type, value). The specification of T service properties is

equal to common QoS specifications.

- *Module properties* (denoted e_M) specifications comprise tuples of types and functions: $e_M = <m_1, ..., m_k>$ and m_i = (type, function). The central idea of module property specification is to describe the influence of a module on offered QoS aspects with mathematical functions. 0-ary functions (i.e., constants) define the service aspect guaranteed by the module (e.g., a CRC16 module guarantees a residual error rate lower than 10^{-9}). N-ary functions describe the influence of the module on the particular attribute. For example, a CRC16 module reduces the offered throughput to a certain percentage depending on the current load of the end system and the offered throughput. Module properties are measured and evaluated before they are integrated in Da CaPo.[1]

- *Application requirements* (denoted e_{AR}) are specified by tuples of attribute types, attribute values and weight function: $e_{AR} = <ar_1, ..., ar_m>$ and ar_i = (type, value | *, weight function). Attribute values in the application requirements specify so-called *knock-out conditions* and indicate that the selected protocol must fulfill these requirements (equal to guaranteed QoS parameters). If there are no threshold values, it is denoted with the *don't care value* "*", that is equal to the least element in ordered value sets. Weight functions serve to define the relative importance of the attribute with respect to other attributes. The obligation of layer C to fulfill the objectives of the application defined by weight functions is weaker than for guaranteed QoS parameters and stronger than for best effort QoS parameters in traditional approaches. L enables us to formulate contradictory requirements (e.g., high bandwidth and low costs are required) and to deal with a wide range of application requirements, mainly introduced by new multimedia applications.

- *Layer C protocol properties* (denoted e_P) are obviously equal to C service properties. Consequently, they are specified analogous to T service properties: $e_{SC} = <sc_1, ..., sc_l>$ and sc_i= (type, value), which are equal to common QoS specifications.

A basic approach to select the best protocol configuration is to estimate the properties of all possible combinations and to compare these properties with the application requirements. The configuration algorithm must retrieve the protocol graphs of the invoked C service from the database. The instantiation of all protocol functions in a protocol graph with one of their modules creates one possible configuration. The resulting graph is called the *module graph*. To estimate the properties of a protocol configuration (denoted e_P) we start with the properties of the T service e_{ST} and calculate the influence of the next higher modules on e_{ST}. Step by step, we calculate the influence of the next higher module in the module graph on the previous results up to the highest module. The result of this process is a property description of the protocol configuration e_P. The unified representation enables a direct comparison of application requirements e_{AR} and protocol properties e_P. If a configured protocol fulfills the application requirements, i.e., does not violate any knock-out conditions, we say the protocol is in compliance with the application requirements: e_{AR} *comp* e_P. We measure the grade of correspondence of e_{AR} and e_P in the *compliance degree*: $cd(e_{AR}, e_P)$, by considering the weight functions of e_{AR}. The protocol with the highest compliance degree is the best configurable protocol with respect to the application requirements. Criterion (C1) formalizes the configuration task:

1. Currently, we are developing a tool to automatically derive the performance related aspects of module properties.

find P such that: $cd(e_{AR}, e_P) \rightarrow max$ subject to: $e_{AR} comp\ e_P$ (C1)

3. Da CaPo architecture

The realization of the three layer model is characterized by three co-operating active entities and a passive database:

- The heuristics method CoRA - which is described in detail in this paper - determines appropriate protocol configurations at runtime.
- The connection management assures that communicating peers use the same protocol for a layer C connection [12]. The connection manager negotiates with its peer a common configuration, initiates protocol establishment and release, and handles errors which cannot be treated inside single modules. Three different negotiation scenarios are maintained and can be selected by application requirements: unilateral, bilateral and combined configuration. Furthermore, the connection manager coordinates the reconfiguration of a protocol if the application requirements are no longer fulfilled.
- The resource manager provides an efficient runtime environment for Da CaPo protocols [13]. The resource manager performs the following tasks: linking and initialization of modules, packet forwarding within protocols, synchronization of parallel modules, monitoring of all protocols and available resources, and release of modules and resources. The monitoring component stores all relevant information in the local database and requests the connection manager to coordinate the protocol reconfiguration if knock-out conditions are offended.
  In contrast to other architectures, which need a process per data packet [14], Da CaPo uses only one process per protocol. Furthermore, it is possible to integrate the application into this process as the application can be designed to have the same interface as the modules. This way, the application can directly benefit from the buffer management provided by the resource manager. Thus, unnecessary copy operations are avoided. On a single processor machine the modules are sequentially executed, while on a multi-processor machine or on a machine with specialized hardware the modules are started in parallel and synchronized by the resource manager [9].
- A database stores information of general interest: protocol graphs, available modules, and module properties. Mainly, CoRA and the resource manager are handling this information.

4. Solving the configuration task

Applications request a layer C connection from Da CaPo by specifying communication partner(s)[2] and application requirements. Consequently, protocol configuration has to be done after the connect request from the application because application requirements and properties of T services which might be used are not known in advance (in particular, in heterogeneous environments). The configuration must be as efficient as possible to keep the connection establishment delay of layer C connections small. But configuration in general and protocol configuration in particular is a complex task. The set of all possible combinations depends on the number of modules per protocol function PF_i (denoted N-PF_i), the number of protocol functions in the protocol graph

2. The current version of Da CaPo supports only one-to-one connections, but we intend to integrate multicast connections in the near future.

PG_j (denoted N-PG_j), and the number of usable T services (denoted N-ST)[3]. Altogether, the configuration process must consider

$$N = \sum_{i=1}^{N\text{-}ST} \left(\prod_{j=1}^{N\text{-}PG_i} N\text{-}PF_j \right)$$

possibilities. Our first approach for an algorithm (we subsequently call this the *base approach*) was based on an exhaustive search of all possible configurations. The results were rather disappointing, as measurements on a NeXT workstation[4] showed that a protocol graph with only two protocol functions took 56 ms to configure, while more reasonable problem sizes immediately caused configuration times in the order of tens of seconds. Thus, the base approach is not suitable for real-time protocol configuration.

In order to decrease configuration time we developed the heuristics method CoRA. The following sections analyse three important aspects of our model with respect to the development of the configuration heuristic:

- protocol graphs and module properties,
- classification of attributes and modules, and
- integration of complex protocol mechanisms.

4.1. Protocol graphs and module properties

The configuration task requires to instantiate protocol functions with appropriate modules. In our base approach, we implicitly judge the suitability of modules by computing their influence onto the offered QoS in order to compare properties of a configuration with application requirements. In other words, only full configurations are explicitly judged and selected. This approach comprises two disadvantages:

- The exhaustive search for the best configuration simply calculates the compliance degree of all possible configurations. Obviously, there is no structure in this approach but efficient search algorithms are generally based on a structure (e.g., breadth-first search or branch-and-bound algorithms are performed on tree structures).
- Module selection is based on the comparison of protocol properties with application requirements, there is no support for explicit judgement and selection of single modules.

The central structure in the configuration task is the protocol graph. Generally, several modules may be available for each protocol function in the protocol graph. Instead of examining all module combinations we sort the modules of a protocol function according to the most relevant aspect of configuration of light-weight protocols: the module weight. In Figure 2, the left protocol graph is associated with the unordered set of available modules while the right protocol graph includes for each protocol function an ordered list of available modules.

The module weight describes the end system load introduced by this module. In current end systems CPU and available bandwidth at network interfaces are major bottlenecks in high-speed

3. The number of different T services is equal to the number of different protocol graphs.

4. Measurements were performed on a NeXTstation with Motorola 68040 processor (25 MHz).

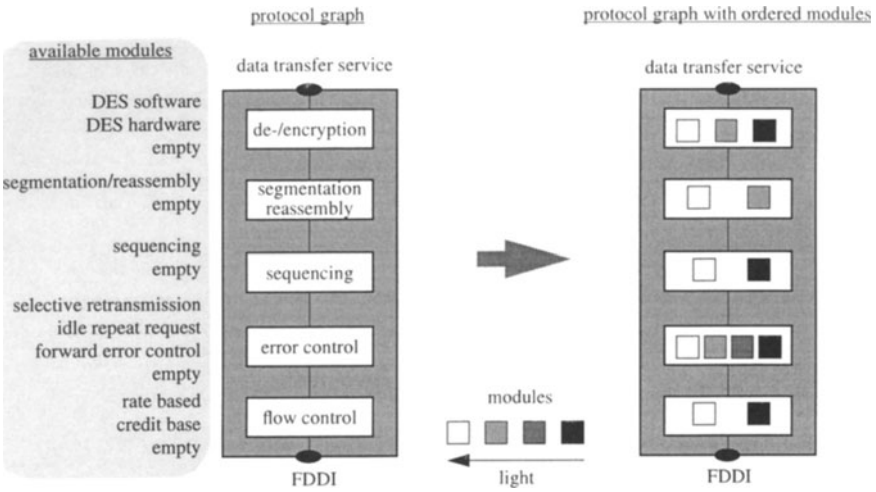

Figure 2. Sorting modules

communications, whereas memory usage is of minor importance. Consequently, we define module weight in terms of

- number of CPU cycles needed to process a packet (denoted CPACKET) independent of the packet length,
- number of CPU cycles per byte (denoted CBYTE) used to process a full packet and
- relative influence on the number of bytes to be transmitted (denoted NET_TRAFFIC). For instance, forward error control modules increase the amount of data because they introduce redundancy (i.e., NET_TRAFFIC > 1), while compression modules decrease the amount of data (i.e., NET_TRAFFIC < 1).

All factors are combined in criterion (C2) to define the module weight:

$$\text{WEIGHT} = \ln(\exp(\text{CBYTE}) + \text{CPACKET})\,\text{NET\_TRAFFIC} \qquad \text{(C2)}$$

The particular combination of CBYTE and CPACKET is used to define the module weight as independent as possible from currently used packet length. The derivation of criterion (C2) is based on several experiments with different packet length and different module properties [15].

In general, there is a linear relation between module weight and performance reduction caused by the module: the higher the module weight the higher the performance reduction. The only exception are modules storing packets for a certain time, e.g., a flow control module stores a packet for a certain time if the current packet rate is too high. The module weight is evaluated off-line and enables us to define a general rule: to configure the lightest protocol means to select the lightest module (i.e., the empty module) for each protocol function.

Nevertheless, configured protocols should not only be as light as possible; it is of major importance to satisfy knock-out conditions and weight functions of the application requirements. In sev-

eral cases a module obviously offends a knock-out condition, consequently, this module can be excluded from the configuration process. For example, if the application demands a high degree of security all modules of the protocol function encryption/decryption which do not fulfill this requirement (e.g., the empty module) need not to be considered further. Each eliminated module reduces the number of possible configurations and thereby decreases configuration time.

The consideration of other knock-out conditions and weight functions which are not corresponding to the weight order of the modules is more complex. Classification of attributes and modules is an approach to reduce this complexity.

4.2. Attribute and module classification

All attribute types in $\mathcal{A}$ may be classified into five groups (see Figure 3), similar to the classification discussed in [16]. The first three groups summarize user aspects and might be considered in application requirements:

- performance related attributes like throughput and delay (denoted PERF($\mathcal{A}$)),
- reliability related attributes like bit error probability, packet loss probability, ordering, and duplication (denoted REL($\mathcal{A}$)), and
- miscellaneous attributes (denoted MISC($\mathcal{A}$)), including all important aspects for the application (e.g., costs, security, data compression, and synchronization) except performance and reliability attributes.

Attributes of the following two groups express layer C aspects and are used within layer C to determine a consistent and suitable configuration:

- resource related attributes (denoted RES($\mathcal{A}$)), including factors of module weight and availability of resources (e.g., CBYTE, CPACKET, and NET_TRAFFIC), and
- protocol related attributes (denoted PROT($\mathcal{A}$)), used to check the consitency of a protocol configuration (e.g., preconditions of modules) and to adapt modules to the current configuration (e.g., maximal transfer unit size and header description).

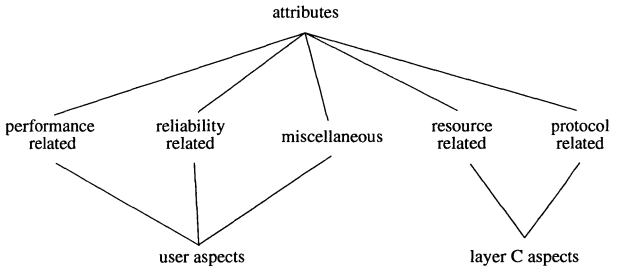

Figure 3. Attribute classification for CoRA

Obviously, all modules influence performance related and resource related attributes (except empty modules). In particular, all modules decrease the performance (except fast hardware compression modules). Consequently, the performance of T services has to be at least equal to the per-

formance required by the application. That means, the selection of a proper T service is of central importance for the configuration of a protocol with sufficient performance. Further performance based selection of modules is influenced by the module weight (i.e., resource related attributes).

Attributes related to reliability issues and miscellaneous issues are influenced by different sets of protocol functions or modules, respectively. The reliability attributes are influenced by modules like CRC, idle repeat request, selective retransmission, and rate based flow control modules. Miscellaneous attributes are influenced by modules performing for instance presentation coding, encryption and decryption, or compression and decompression. These disjoint sets of protocol functions may be independently configured, which in turn drastically decreases the complexity of the entire configuration process. Let us consider a simple example with a protocol graph consisting of four protocol functions, each protocol function can be instantiated by four different modules. In total, there are $4^4 = 256$ possible configurations. Assuming that two protocol functions only affect reliability attributes and two protocol functions only affect miscellaneous attributes there are $4^2 + 4^2 = 32$ possibilities (i.e., only 12.5 percent of all configurations need to be investigated).

4.3. Integrating complex protocol mechanisms

In our model, a protocol function can be realized by different protocol mechanisms, i.e., protocol functions and protocol mechanisms are in a (1:n)-relation. In practice, there are several well known protocol mechanisms, each realizing multiple protocol functions, i.e., a (m:n)-relation between protocol functions and mechanisms. For example, the protocol mechanism "idle repeat request" performs the protocol functions "flow control", "packet loss detection", "packet loss correction", and "resequencing". This (m:n)-relation is contradicting to the simple abstraction hierarchy (protocol function - protocol mechanisms - modules) in our model.

To support protocol functions of any granularity and to integrate complex protocol mechanisms we extended our basic model by introducing one-level nodes (i.e., protocol functions) and two-level nodes in protocol graphs. Two-level nodes may be instantiated by sub-graphs consisting of a main protocol mechanism (which might be empty) and protocol functions. Additional protocol functions are combined with the main mechanism to specify the particular pre- and post-processing functionality. Protocol graphs as well as sub-graphs are defined by protocol engineers. Figure 4 illustrates several sub-graphs for the two-level node "reliability". The main mechanism "forward error control" requires ordered packet sequences, consequently its pre-processing part contains the protocol function "resequencing". In contrast, the main protocol mechanism "selective retransmission" supports no packet resequencing, therefore its post-processing part contains the protocol function "resequencing".

5. CoRA

All concepts discussed in the previous section are combined in the heuristic CoRA. CoRA consists of six steps:

(1) pre-decision,

(2) module elimination,

(3) T service selection,

(4) configuration,

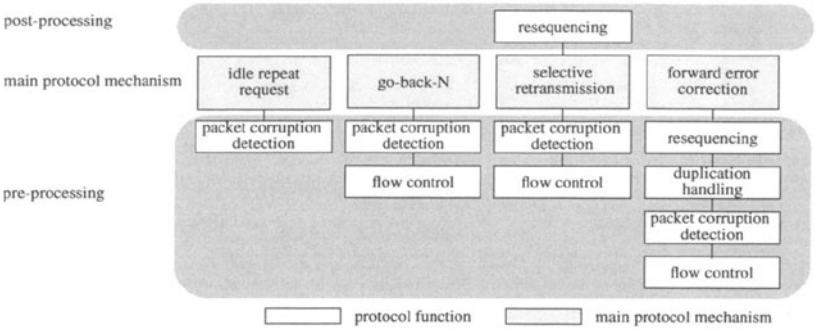

Figure 4. Sub-graphs of two-level node reliability

(5) optimization, and

(6) post-decision.

No single step guarantees to find a suitable configuration, but the combination of these steps within the modes FIRST, MINI, MEDIUM, and FULL guarantees to find at least a configuration (if it exists) which is in compliance with the application requirements. In mode FIRST, CoRA stops after finding the first configuration (step(4)) which is in compliance with the application requirements. In the other modes the steps (3), (4), and (5) are performed at least once to increase the compliance degree.

5.1. Pre-decision

The first step of CoRA is the pre-decision to determine whether it is currently appropriate to configure and establish a new protocol or not. It might be impossible to configure a suitable protocol because of high end system load or fully utilized network interfaces. Furthermore, after introducing a new layer C connection (i.e., additional end system load) all established connections should fulfill the application requirements. The decision is based on three factors:

- CPU_LOAD denotes the relative load of the CPU during the last time interval. Values of CPU_LOAD are in the range [0..1]. The value zero indicates an idle situation and value one full load on all CPUs, i.e., the higher CPU_LOAD the lower the probability to establish a new layer C connection without offending knock-out conditions of established connections.
- NET_STATE summarizes the current utilization of network interfaces (respectively, T services). Values of NET_STATE are in the range [0..1]. The value of NET_STATE is equal to zero if all network interfaces are idle and equal one if they are fully utilized. The higher NET_STATE the lower the probability to establish a new layer C connection without offending knock-out conditions of established connections.
- COMP serves to estimate the average distance between properties of all established layer C connections and the corresponding knock-out conditions. If the properties of established layer C connections are much better than demanded in the application requirements (i.e., knock-out

conditions) then the probability is high that the properties of all connections will remain in compliance with the corresponding application requirements. In contrast, if the distance between knock-out values and properties of connections is low, then the probability of offending knock-out conditions by establishing a new layer C connection is high. Values of COMP are in the range [0..1]. The higher COMP the higher the average distance and the higher the probability to introduce a new layer C connection without offending knock-out conditions of established connections.

The combination of these three factors has to be lower than a threshold value named PRE_THRESHOLD. The threshold value PRE_THRESHOLD has to be determined according to the resource allocation strategy performed in the particular end system (e.g. by a system administrator). The higher PRE_THRESHOLD the more layer C connections may be established at the same time. Criterion (C3) merges CPU_LOAD, NET_STATE and COMP according to their positive respectively negative influence:

$$\text{PRE\_THRESHOLD} > \frac{\text{CPU\_LOAD} + \text{NET\_STATE}}{2}(1 - \text{COMP}). \qquad \text{(C3)}$$

If criterion (C3) is not fulfilled CoRA terminates and indicates that it is currently impossible to configure a proper protocol and to establish the corresponding connection.

5.2. Elimination

The aim of the module elimination is to exclude as many modules as possible at the start of the protocol configuration to decrease its complexity. Candidates to be excluded are:

- Modules that are currently unavailable (e.g., hardware modules) and T services that are currently fully utilized.
- Modules whose weight is too high to run under the current end system load.
- Modules whose costs are higher than tolerated in the application requirements.
- Modules of the miscellaneous class which obviously do not fulfill the application requirements. For example, the application requires the presentation coding mechanism eXternal Data Representation (XDR). All modules - except the XDR module - can be excluded from the search.

5.3. T Service Selection

Generally, there is one protocol graph defined for each supported T service. The step T service selection serves to order the T services (i.e., protocol graphs) and to concentrate on the most promising one. Primarily, performance related attributes are considered to order the T services, because layer C protocols generally decrease the performance offered by a T service. Consequently, the performance of the T service must be at least in compliance with the application requirements to find a suitable configuration, i.e., $\text{PERF}(e_{ST}) \geq \text{PERF}(e_{AR})$. Additionally, the T service should correspond to the weight functions in the application requirements, i.e., $cd(e_{AR}, e_{ST}) \rightarrow \max$. From the systems point of view, resources should be economically allocated and the difference between application requirements and T service properties (responding to network resources to be allocated) $|e_{AR} - e_{ST}|$ should be as small as possible. Criterion (C4) combines these aspects to select a T service:

$$\text{find ST such that: } \frac{cd(e_{AR}, e_{ST})}{|e_{AR} - e_{ST}|} \rightarrow \max \qquad \text{subject to: } \frac{PERF(e_{ST})}{REDUCT} \geq PERF(e_{AR}). \tag{C4}$$

The parameter REDUCT estimates the relative performance reduction of the layer C protocol to be configured.

5.4. Configuration

Step (4) looks for a configuration that is in compliance with the application requirements. Consequently, only knock-out conditions comprising attributes of the classes REL($\mathcal{A}$), PERF($\mathcal{A}$), and MISC($\mathcal{A}$) have to be considered. Step (4) selects a module for all nodes in one of these classes. The three classes are independently processed to decrease complexity.

The first task is to generate a one-level protocol graph out of the protocol graph determined in the previous step. Two-level nodes are expanded by selecting a sub-graph. The selection is based on the subsequently described judgement of the main protocol mechanisms (denoted MPM) in the sub-graphs. The compliance degree of the protocol mechanism MPM on top of the selected T service (ST) should be as high as possible, i.e., $cd(e_{AR}, e_{ST-MPM}) \rightarrow \max$. Furthermore, the performance of this configuration must be higher than required by the application, and the number of unresolved pre-conditions of the protocol mechanism should be small. Criterion (C5) combines these aspects to select a sub-graph:

$$\text{find MPM such that: } \frac{cd(e_{AR}, e_{ST-MPM})}{\#\text{unresolved Preconditions} + 1} \rightarrow \max$$

$$\text{subject to: } PERF(e_{ST-MPM}) \text{ comp } PERF(e_{AR}). \tag{C5}$$

The second task is to instantiate each node in a one-level protocol graph with a module such that the resulting configuration is in compliance with the application requirements. We start with the lowest node of the protocol graph, consider application requirements and preconditions of higher nodes and try to fulfill them by selecting a suitable module (according to the weight order). If a single module cannot fulfill the requirements, we look for higher modules influencing the same attribute and examine the different combinations. If no module or module combination can be found, we start a further iteration of step configuration and generate a further one-level protocol graph. Step (4) terminates after the first configuration is found that is in compliance with the application requirements.

5.5. Optimization

The optimization step attempts to improve the protocol configuration (i.e., its compliance degree) determined in step (4). We order the tuples of the application requirements according to their importance for the application to purposefully increase the compliance degree. Two aspects have to be considered to define the importance of the application requirement tuples. First, the value of the weight function applied to the minimal required value (i.e., knock-out value) and second the distance between the knock-out value and the value offered from the currently examined protocol configuration. This protocol configuration is in compliance with the application requirements. Consequently, the lower the distance between current value and required value the higher the probability to increase the compliance degree. By placing the distance in the denominator of criterion (C6) we prefer application requirements that are weakly fulfilled[5]:

$$IMPORTANCE = \left| \frac{required\_value}{current\_value - required\_value} \right| weight(required\_value) \tag{C6}$$

Step (5) takes the most important tuple and tries to increase the compliance degree for this attribute by examining all modules influencing this attribute. After improving the compliance degree, all tuples are ordered again and the optimization step tries to improve the most important attribute. If the optimization of one attribute is not possible, the next attribute (according to the order) will be examined. This is done until all attributes are examined and no further improvement is possible.

5.6. Post-decision

The post-decision step serves to decide whether it is possible to establish a connection for the selected protocol without decreasing the performance of existing connections too much. In contrast to step (1) the weight of the particular configuration is now known (denoted W_P) and a more precise decision can be taken. We compare the relative utilization of the end system (measured in CPU_LOAD and NET_STATE, see step (1)) with the relative protocol weight instead of the threshold value PRE_THRESHOLD. The relative protocol weight is given by the relation between the protocol weight W_P and the protocol weight W_{max}. W_{max} denotes the weight of a theoretical protocol that would fully utilize the end system, i.e., the weight of the heaviest protocol. The post-decision step allows the establishment of a new protocol if criterion (C7) holds:

$$\frac{W_P}{W_{max}} \leq \frac{CPU\_LOAD + NET\_STATE}{2} \qquad \text{(C7)}$$

5.7. Modes

The stepwise approach of CoRA supports four different modes with increasing computational complexity and improved results:

- FIRST: Mode FIRST returns the first configuration which is found in step (4).
- MINI: Mode MINI operates on a one-level protocol graph, that is determined in step (4), and looks for a configuration of this one-level protocol graph with the highest compliance degree in step (5). In other words, mode MINI extends mode FIRST by additionally performing step (5).
- MEDIUM: Mode MEDIUM operates on a two-level protocol graph, that is determined in step (3), and examines all sub-graphs, in contrast to mode MINI. In other words, step (4) and (5) are performed several times.
- FULL: Mode FULL considers all two-level protocol graphs and all possible sub-graphs to find the configuration with the highest compliance degree. The steps (3), (4) and (5) are performed multiple times in mode FULL.

6. Performance evaluation

We implemented CoRA in ANSI C such that all criteria documented in this paper (C1) - (C7) might be changed easily. This enables us to later adapt CoRA based on experiences. In order to evaluate the performance of CoRA we elaborated several sample scenarios. The values used in our

5. In case that current_value is equal to required_value we simply assign a very high importance to the application requirement. By this we avoid a division by zero and indicate that there is a high probability to improve the compliance degree with this application requirement.

scenarios to specify application requirements, properties of layer T connections, and module properties are derived from the literature, our measurements and estimations. We compared the compliance degree of CoRA results with the maximum compliance degree (result of the base configuration approach) to determine the quality of CoRA results (measured in percentage of the maximum compliance degree). All measurements are performed on a Sun SparcStation 10/30.

Generally, the performance of CoRA depends on two aspects: first, the complexity of the problem (measured in number of possible configurations) and second, the structure of the particular configuration task. Obviously, the configuration task is determined by the application requirements. Restrictive knock-out conditions are applied in CoRA for module elimination and influence the performance of CoRA in two ways. In mode FIRST and mode MINI it is harder to find a configuration which is in compliance than in scenarios with non-restrictive application requirements (i.e., knock-out conditions that could be easily fulfilled). However, mode MEDIUM and mode FULL benefit from the module elimination based on restrictive knock-out conditions. This behavior is documented in the results of a sample scenario with 1,876,896 possibilities (Table 1). The scenario comprises two protocol graphs including the protocol functions "monitoring", "compression", "security", "presentation coding", and the two-level node "reliability" that are defined on the T services "IP" and "ATM/AAL5". Several modules with different properties are available for each protocol function. Application requirements 1 (AR-1) comprise no hard requirements, e.g., the performance requirements could be fulfilled by both T services. In contrast, the application requirements 2 (AR-2) include restrictive knock-out conditions on the attributes "throughput", "delay jitter", and "packet loss". The throughput and delay jitter requirements could only be fulfilled by the T service "ATM/AAL5". The base approach needs more than 12 minutes to solve the problems. Table 1 compares the configuration times of CoRA for AR-1 and AR-2 applied to the same protocol graphs and set of modules.

Table 1. Non-restrictive versus restrictive application requirements

	FIRST	MINI	MEDIUM	FULL
AR-1	12 ms; Quality 83%	19 ms; Quality 83%	79 ms; Quality 88%	170 ms; Quality 100%
AR-2	13 ms; Quality 91%	22 ms; Quality 98%	49 ms; Quality 100%	100 ms; Quality 100%

We examined the relation between configuration times and complexity of the configuration task with several hundred different measurements in four different basic scenarios (denoted A, B, C, and D). Each scenario is based on some protocol graphs and a set of available modules, that determine the complexity of the configuration task. Scenario A comprises 175,959 possible configurations, scenario B 1,876,896 possibilities, scenario C 7,288,848 possibilities and scenario D 58,560,768 possibilities. Within each scenario we performed measurements with different application requirements. The configuration times of the base approach directly depend on the complexity: it needs approximately 50 seconds in scenario A, 12 minutes in B, 28 minutes in C, and 3.5 hours in scenario D.

The graphs in Figure 5 represent the average configuration times for the four CoRA modes as well as the average qualities of the results (denoted Q) in the modes FIRST, MINI, MEDIUM, and FULL. The maximal standard derivation of the configuration times is nearly 50%, caused by the different application requirements. These deviations show the same behavior of CoRA than discussed in the first sample scenario (Table 1). However, the maximum standard derivation of the quality of the results is below 15%. Obviously, the scenario A with the lowest complexity needs in most modes the longest configuration times. This demonstrates that the response times of CoRA

depend more on the particular problem structure (e.g., number of T services, number of two-level nodes, properties of modules and T services, and number of knock-out conditions and weight functions) than on the number of possible configurations.

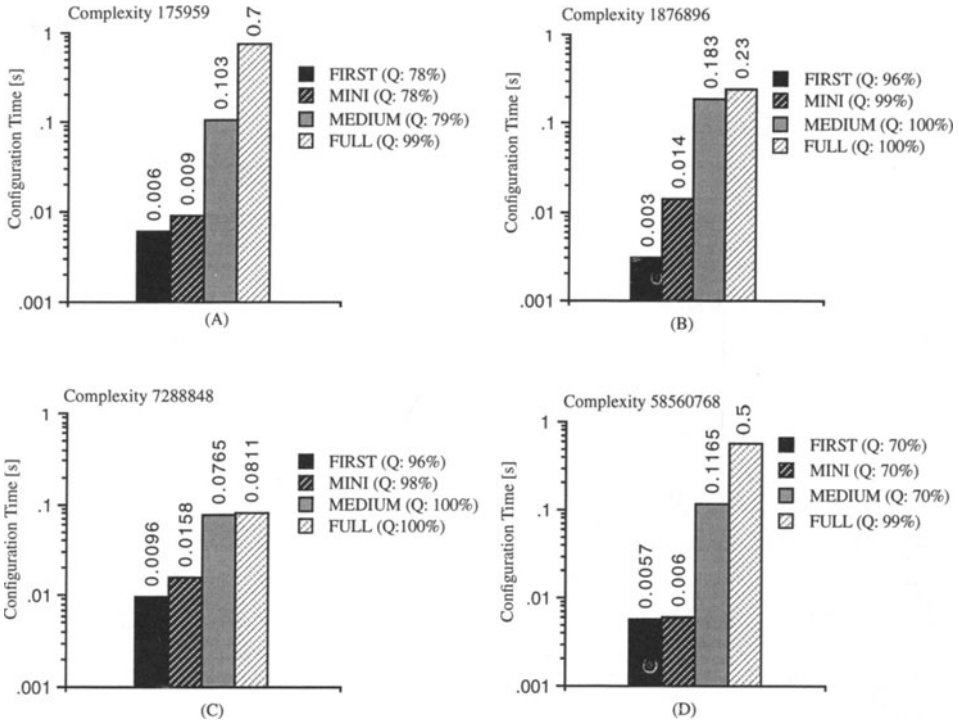

Figure 5. Average configuration times

Summarizing the results presented in Figure 5, we recognize that mode FIRST determines configurations with 70% and more of the maximum compliance degree within less than 10 milliseconds. Furthermore, the quality of results in modes MEDIUM and FULL is in most cases equal to the optimal configuration (i.e., 100% quality), while the configuration times in mode MEDIUM are less than 200 milliseconds and in mode FULL less than 500 milliseconds in most scenarios.

7. Conclusions

In this paper, we presented the concepts and implementation of the configuration heuristic CoRA applied within the Da CaPo project. The definition of module weight is used twice in CoRA; on the one hand it speeds-up the configuration process and on the other hand it supports careful resource allocation. A further speed-up of the configuration is possible by classifying

attributes, protocol functions, and modules. The integration of complex protocol mechanisms enables us to study fine granular protocol mechanisms as well as complex mechanisms and to compare them in the same environment. Measurements demonstrate the high performance and quality of CoRA. CoRA enables us to configure protocols at runtime, only marginally increasing connection establishment delay with respect to connection establishment delay of fixed protocols.

Our future work is to extend the set of modules and to derive their properties. The implementation of multimedia applications on top of Da CaPo will be used to evaluate the performance and to demonstrate the feasibility of CoRA and the entire Da CaPo system in the context of "real-life" problems. Furthermore, the practical experiences should be used to improve and refine the configuration and particularly the resource allocation in CoRA.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments. We want also to thank all the Da CaPo team and all students that made contributions to Da CaPo, namely Marcel Dasen, Peter Imhof, Alireza Olumni, Mahan Satari, Martin Vogt, Janusch Waclawczyk, Thomas Walter, and Thomas Ward. Finally, we would like to thank Arlette Gaillard for a fruitful discussion.

References

[1] Harris, D.: "Streams", in: Kochanan, S.G., Wood, P.H. (Editors): "UNIX Networking", pp. 133-170

[2] Haas, Z.: "A Communication Architecture for High-speed Networking", in: Proceedings of IEEE INFOCOMM'90, 9th Annual Joint Conference of the IEEE Computer and Communications Societies, Los Almos, California; Vol. 2, June 1990, pp. 433-441.

[3] O'Malley, S.W., Peterson, L.L.: "A Dynamic Network Architecture", in: ACM Transactions on Computer Systems, Vol. 10, No. 2, May 1992, pp. 110-143

[4] Metzler, B., Miloucheva, I.: "Specification of the Broadband Transport Protocol XTPX", CEC Deliverable Number: R2060/TUB/CIO/DS/P/001/b2, February 1992

[5] Campbell, A., Coulson, G., Gracia, F., Hutchinson, D., Leopold, H.: "Integrated Quality of Service for Multimedia Communications", in: Proceedings of IEEE INFOCOMM'93, San Francisco, March 1993

[6] Box, D. F., Schmidt, D. C., Tatsuya, S.: "ADAPTIVE - An Object-Oriented Framework for Flexible and Adaptive Communication Protocols", in: Proceedings hpn92, 4th IFIP Conference on High Performance Networking, December 1992

[7] Zitterbart, M., Stiller, B., Tantawy, A.M.: "Application-Driven Flexible Protocol Configuration", in: IEEE Journal on Selected Areas in Communications, Vol. 11, No. 4, May 1993, pp. 507-518

[8] Rzehak, R.: "In Discussion: Mrs. Zitterbarts Contribution to KiVS 93" (in German), PIK Praxis der Informationsverarbeitung und Kommunikation, 2/93

[9] Vogt, M., Plagemann, T., Plattner, B., Walter, T.: "Parallelism Aspects in Da CaPo" (in German), GI Workshop on Architecture and Implementation of High-Performance Communication Systems, Karlsruhe, January 1994

[10] Plagemann, T., Plattner, B., Vogt, M., Walter, T.,: "A Model for Dynamic Configuration of Light-Weight Protocols", in: Proceedings IEEE Third Workshop on Future Trends of Distributed Computing Systems, Taipei, Taiwan, April 1992, pp. 100-110

[11] Plagemann, T., Plattner, B., Vogt, M., Walter, T.: "Modules as Building Blocks for Protocol Configuration", Proceedings of International Conference on Network Protocols ICNP'93, San Francisco, October 1993, pp. 106-113

[12] Plagemann. T., Waclawczyk, J., Plattner, B.: "Management of Configurable Protocols for Multimedia Requirements", to appear in: Proceedings of First ISMM International Conference on Distributed Multimedia Systems and Applications, Honolulu, August 1994

[13] Vogt, M., Plagemann, T., Plattner, B., Walter, T.: "A Runtime Environment for Da CaPo", in: Proceedings of INET'93, International Networking Conference Internet Society, San Francisco, August 1993

[14] Hutchinson, N. C., Peterson, L. L.: "The x-Kernel: An Architecture for Implementing Network Protocols" in: IEEE Transactions on Software Engineering, January 1991, pp. 64-76

[15] Gotti, A: "CoRA - Configuration and Resource Allocation in Da CaPo" (in German), Diploma Thesis at Computer Engineering and Networks Laboratory, ETH Zurich, March 1994

[16] ITU - Telecommunications Standardization Sector, Study Group 7 / WP: "Quality of Service Framework (draft no. 3)", Source: SC 21/WG (N 1298), Geneva, February 1994