ADAPTIVE

A Flexible and Adaptive Transport System Architecture to Support Multimedia Applications on High-Speed Networks

Technical Report 92-46

Douglas C. Schmidt, Donald F. Box, and Tatsuya Suda
schmidt@ics.uci.edu, dbox@ics.uci.edu, and suda@ics.uci.edu
Department of Information and Computer Science,
University of California, Irvine,
Irvine, CA 92717, U.S.A.
(714) 856-4105 (phone)
(714) 856-4056 (fax) [1]

[1]This material is based upon work supported by the National Science Foundation under Grant No. NCR-8907909. This research is also in part supported by University of California MICRO program.

Abstract

This paper describes a transport system architecture called ADAPTIVE that we are developing to support multimedia applications utilizing high-speed networks. ADAPTIVE stands for "A Dynamically Assembled Protocol Transformation, Integration, and Validation Environment." It is a transformation-based system with the goal of providing policies and mechanisms for interactively *specifying* and *configuring* a flexible and adaptive transport system. In addition, it aims to provide a controlled prototyping environment for *monitoring*, *analyzing*, and *experimenting* with the effects of different transport system designs and implementations on application performance.

1 Introduction

The performance of distributed applications (such as remote terminal access, network file servers, distributed databases, collaborative work activities, bulk data transfer, computer imaging, audio- and video-conferencing, full-motion video, scientific computation and visualization) is influenced by network factors and transport system[1] factors. Several key network factors include channel speed, bit-error rate, and congestion levels at the intermediate switching nodes (ISNs). Transport system factors include both protocol processing activities (such as connection management, transmission control, and error protection) and general operating system hardware and software factors (such as memory latency, CPU speed, interrupt and context switching overhead, process architecture, and message buffering).

Advances in fiber optics and VLSI technology have increased network channel speed by several orders of magnitude [1]. Coupled with the increase in application and network diversity described in Section 2, these changes are shifting performance bottlenecks from the network factors to the transport system factors. Therefore, transport system architectures must be more *flexible* and *adaptive* to handle the diversity and dynamism in network characteristics and application communication requirements [2]. However, existing transport systems, which typically offer only a small number of monolithic protocols, are unable to meet these flexible and adaptive requirements adequately [3].

This paper is organized as follows: in Section 2 we define four important research problems involving transport systems and describe how existing systems do not adequately solve these problems, Section 3 introduces the ADAPTIVE system and outlines its design, and Section 4 summarizes the paper and describes future work.

2 Research Problems

Our research is investigating solutions to the following four problems involving transport systems:

(A) The Throughput Preservation Problem: As noted by [1, 4, 5], only a limited fraction of the available bandwidth in high-speed networks is being delivered to applications. This situation results from transport system overhead not decreasing as rapidly as the network channel-speed is increasing. The overhead results from activities such as memory-to-memory copying and process management operations like interrupt handling, context switching, and scheduling.

(B) The Multimedia Application Diversity and Dynamism Problem: Another issue (which is related to the throughput preservation problem) is how to support the communication requirements of distributed multimedia applications[2] that are more demanding and dynamic than those found in traditional data applications such as remote terminal access, file transfer, and electronic mail. For example, multimedia applications involve combinations of requirements such as extremely high throughput (*e.g.*, full-motion video), strict real-time delivery (*e.g.*, manufacturing control systems), low latency (*e.g.*, on-line transaction processing), low delay jitter (*e.g.*, voice conversation), multicast capability (*e.g.*, collaborative work activities), high-reliability (*e.g.*, bulk data transfer), temporal synchronization (*e.g.*, audio- and video-conferencing), and some degree of loss tolerance (*e.g.*, hierarchically-coded voice and video). Moreover, multimedia applications impose different network traffic patterns. For instance, some applications generate highly bursty traffic (*e.g.*, variable bit-rate video applications), some generate continuous traffic (*e.g.*, constant bit-rate video applications), and others generate short, interactive, transaction-oriented traffic (*e.g.*, network file systems using remote

[1]Transport systems are composed of both general-purpose operating systems (such as UNIX and Mach) and the protocols (such as TCP, TP4, XTP, VMTP, and Delta-*t*) that utilize the OS services. In particular, transport systems provide services to applications that encompass more functionality than that provided by the transport layer.

[2]We will refer to these as simply "application requirements" from now on.

Transport Service Class	Example Applications	Average Throughput	Burst Factor	Delay Sens.	Jitter Sens.	Order Sens.	Loss Toler.	Multicast
Interactive Isochronous	Voice Conversation	low	low	high	high	low	moderate	no
	Tele-conferencing	moderate	moderate	high	high	low	moderate	yes
Distributional Isochronous	Full-Motion Video	very high	high(*)	high	high(*)	low	moderate	yes
Real-Time Non-Isochronous	Manufacturing Control	moderate	moderate	high	somewhat	variable	low	yes
Non-Real-Time Non-Isochronous	File Transfer	moderate	low	low	low	high	low	no
	Remote Login	very low	high	high	low	high	low	no
	On-Line Transaction Processing	low	high	high	low	variable	low	no

Table 1: Application Transport Service Classes

procedure calls (RPC)). In addition, application requirements may change dynamically (*e.g.*, consider an audio-conferencing application that switches between unicast and multicast as participants join and leave the conversation). Table 1 illustrates the diversity of transport requirements for several representative classes of distributed applications [6].[3]

(C) The Network Diversity and Dynamism Problem: Network diversity involves both the static architecture and dynamic state of the underlying network (or more generally, the internetwork). For example, different network characteristics include *diameter* (*e.g.*, LANs, MANs, and WANs), *channel speeds* (*e.g.*, Token Ring (4/16 Mbps), Ethernet (10 Mbps), FDDI (100 Mbps), and ATM (155/622 Mbps)), *channel bit-error rate probability* (*e.g.*, approximately 10^{-4} for copper vs 10^{-9} for fiber), *network service type* (*e.g.*, datagram vs. virtual circuit), *media access control* (*e.g.*, CSMA/CD vs. token passing), and maximum transmission unit (*e.g.*, 4,500 bytes for an FDDI frame vs. 1,500 bytes for Ethernet packets vs. 48 bytes for ATM cells). However, applications may continue to interoperate between three typical network environments, including (1) low-utilization, low-latency LANs (*e.g.*, Ethernet), (2) congestion-prone, high latency WANs (*e.g.*, the current Internet), and (3) high-bandwidth, high latency WANs (*e.g.*, ATM-based B-ISDN public access networks) [3].[4] Problems arising from handling this diversity and dynamism involve determining appropriate end-to-end congestion and error protection schemes that effectively utilize the high bandwidth channels and adapt quickly to dynamically changing network conditions (such as congestion).

(D) The Communication Software Development Complexity Problem: Communication software has traditionally been difficult to develop, due in part to the complexity arising from trying to simultaneously maximize performance, functionality, and portability across heterogeneous operating environments [7]. This heterogeneity is reflected in the complexity of software architectures that must support (1) diverse applications communicating via (2) diverse transport systems that interoperate over (3) diverse network environments [8]. To effectively support this network and application diversity, it is increasingly important to structure communication software in a manner that is maintainable, flexible, extensible, and efficient.

[3]Note, the "values" in the table cells are rough approximations; items marked with a (*) are coding dependent.

[4]Network diversity and dynamism are problems that will not completely disappear even if there were only one standardized network infrastructure. In this case, there would still be considerable diversity in network diameter and dynamism in network congestion characteristics.

2.1 Inadequacy of Existing Transport System Solutions

Existing transport systems do not provide effective solutions for all four problems described above. This section describes several reasons for the inadequacy of existing systems.

(A) High Transport System Overhead: Effectively supporting multimedia applications requires suitable levels of performance from both the network as well as from the transport system [9]. However, most general-purpose operating systems do not provide adequate support for transport system activities such as real-time process scheduling, interrupt handling and context switching, message buffer management, layer-to-layer flow control, and multiplexing and demultiplexing [10]. Therefore, due to the transport system overhead, the available bandwidth actually delivered to applications in a high-speed network is reduced by several orders of magnitude [5]. Moreover, this throughput preservation problem persists despite an increase in CPU speeds. There are several explanations for this: (1) networks have increased by 5 or 6 orders of magnitude (*i.e.*, from kbps to Gbps), whereas CPU speeds have only increased by 2 or 3 orders of magnitude (*i.e.*, from 1 MIP up to 100 MIPS) [1], (2) network host interfaces typically generate interrupts for every transmitted and received packet, which lead to increased CPU context switching overhead [1, 11], and (3) despite increasing total MIPS, RISC architectures (such as the SPARC) penalize this interrupt-driven network communication, since they typically exhibit high context switch overhead, resulting from the cost of flushing instruction and data caches and pipelines, storing and retrieving large register windows, etc. [12].

(B) Inflexible and Non-Adaptive Transport System Architectures: Another reason why existing transport systems are unable to meet the demands of diverse applications and networks are that they typically provide only a fixed, statically configured suite of protocols [3]. For instance, it is not possible to dynamically tailor most general-purpose protocols to account for specific application requirements or network characteristics [13]. Moreover, communication software tends to be very inflexible and difficult to modify [3], and is often written in a manner that tightly couples it to a particular transport system environment (*e.g.*, TCP/IP on BSD UNIX) [10, 14]. These factors increase the difficulty of maintaining, extending, and adapting the software to support diversity in networks and applications.

To illustrate the variety of transport system architectures, we distinguish them below by how capable they are of adapting their configurations to support changes in application requirements and network characteristics:

1. *Non-adaptive* transport systems are statically configured at operating system boot time. In this case, the transport system often provides protocol support based upon general usage assumptions such as *unreliable datagram service* (*e.g.*, UDP) versus *reliable byte stream service* (*e.g.*, TCP). Many well-known transport systems belong to this category [15, 16, 17, 18, 19].
2. *Semi-adaptive* transport systems permit postponing complete configuration until connection establishment time. Such transport systems may be configured by negotiating with local and remote hosts and the network. This negotiation process specifically tailors the transport system to account for the application requirements and the available host and network resources such as buffer space, CPU load, virtual circuits, and congestion. Only a handful of transport systems provide any support at all for semi-adaptive configuration of their protocol suites [3, 20, 21]. Furthermore, this support typically involves only operations on the local hosts (*i.e.*, network and remote hosts conditions are *not* considered in the configuration process).
3. *Fully-adaptive* transport systems extend semi-adaptive systems by allowing reconfiguration during the data-transfer phase (*e.g.*, to accommodate changes in application requirements, transport system conditions[5], or network characteristics). Very few transport systems even attempt to provide any sup-

[5]These conditions involve changes in transport system resources such as buffer space, processor load, and available communication ports.

port for fully adaptive reconfiguration, and those that do are experimental research projects [6, 22].

Given the increasing diversity of application requirements and network characteristics (combined with the fact that both may change dynamically), it appears that semi-adaptive and fully-adaptive transport systems may support application/network pairings more effectively than non-adaptive systems.

(C) No Explicit Support for Multimedia Applications and High-Speed, Multi-Service Networks: Many popular transport systems were designed and implemented before the new generation of multimedia applications and high-speed, multi-service networks became widely available. For example, the BSD 4.3 UNIX TCP/IP implementation was originally designed for traditional data applications that require 100 percent error-free transmission over low-bandwidth, high-error network links (*e.g.*, ARPANET). Therefore, TCP/IP (and in many cases the ISO TP[0-4] transport protocols) may provide inadequate support for modern application requirements by not incorporating efficient header[6], fast connection establishment, multicast, security, or quality of service parameters such as prioritized real-time guarantees and application-selectable levels of loss-tolerance [2, 24]. Moreover, many existing protocols suites were designed for low-speed, unreliable, congestion-prone, datagram networks, rather than high-speed, congestion-controlled, virtual-circuit networks such as ATM [25]. For example, the TCP/IP suite does not provide explicit access control[7], rate control, selective retransmission, or support for large flow control windows. Although extensions to TCP and IP have been proposed that address these limitations, these proposals have not yet been integrated into the base standards requirements.

2.2 Related Work

A growing number of projects address flexible and adaptive protocols and architectures for high-speed transport systems.[8] The ADAPTIVE system is primarily influenced by the Programmable Network Prototyping System (PNPS) [29], the *x*-kernel/Avoca projects [3, 10], and the Multi-Stream Protocol (MSP) [22]. PNPS is a flexible environment for prototyping and experimenting with hardware implementations of MAC-layer protocols. ADAPTIVE, on the other hand, focuses on prototyping and experimenting with flexible and adaptive software architectures for higher-layer protocols.

The *x*-kernel is a communications-oriented operating system kernel that supports protocol development and experimentation. It provides a "protocol backplane" consisting of uniform interfaces for reusable communications services such as message handling, multiplexing and demultiplexing, and event management. Relationships between entities in protocol suites are described and implemented via dynamic, modular, and highly-layered protocol graphs[9] composed from virtual- and micro-protocols. Avoca uses *x*-kernel as a run-time environment to support protocol implementation and experimentation. It focuses primarily on flexible implementations of protocols like RPC, UDP, and TCP that support traditional data applications running on traditional networks. ADAPTIVE, on the other hand, focuses on flexible and adaptive transport systems that support multimedia applications running on high-speed networks.

MSP ("Multi-Stream Protocol") is a "feature-rich" transport protocol designed to execute in parallel. In addition, MSP permits protocol configurations to change their mechanisms "on-the-fly" without data loss (*e.g.*, allowing the retransmission implementation to switch from *go-back-n* to *selective repeat* within an active connection). Like the *x*-kernel, MSP focuses more on mechanisms (*i.e.*, *how* to implement the changes) rather

[6]For example, TCP and TP4's checksum is not in the trailer, which precludes simultaneous transmission and checksum computation [23]. Moreover, many fields in the TCP header are not word-aligned and the option formats are not fixed-sized.

[7]Note, TCP's *slow start* and *multiplicative decrease* algorithms are used to simulate access control.

[8]In addition to the research described in this section, other projects focusing on various aspects of flexible and adaptive transport systems and protocols include [1, 21, 23, 24, 26, 27, 28].

[9]A protocol graph is a generalization of a protocol stack; it represents the hierarchical relations between protocols in one or more protocol suites [3].

than on policies (*i.e.*, *when* to make the changes and *what* changes should occur). ADAPTIVE, on the other hand, focuses on both policies and mechanisms (as described in Section 3.1).

3 The ADAPTIVE System

The following section introduces the ADAPTIVE system and describes its distinctive features, compares it with alternative techniques, and outlines the architectural design of its major subsystems.

3.1 ADAPTIVE Overview

ADAPTIVE is "A Dynamically Assembled Protocol Transformation, Integration, and Validation[10] Environment" used to specify, configure, experiment with, and analyze alternative transport system designs and implementations. We are developing ADAPTIVE to help ameliorate the inadequacies of existing systems described in Section 2.1 above. ADAPTIVE is distinguished by its integrated framework that exhibits: (1) support for application and network diversity and dynamism via a *flexible* and *adaptive* architecture, (2) reduction in transport system overhead, (3) focus on policies and mechanisms, and (4) feedback-guided monitoring and measurement. This section briefly describes these aspects of ADAPTIVE.

(A) Support for Both Multimedia Application Requirements and Network Characteristics via a Flexible and Adaptive Architecture: ADAPTIVE provides services that flexibly configure and adaptively reconfigure transport systems based upon network characteristics and communication requirements of applications.[11] It employs a library of reusable transport system mechanisms (*e.g.*, buffer and event managers, connection management, error protection, and transmission control) to facilitate the synthesis of efficient transport systems. Flexibility is an important design criteria since it supports:

- *Prototyping* - Compared with analysis and simulation, prototyping often facilitates more realistic performance results, since it is able to detect and account for interactions between system components [29].
- *Experimentation* - Experimentation is important since there is no clear consensus on the best methods for developing transport systems that support network and application diversity.
- *Diversity* - Transport systems of the future must operate effectively in both a standardized high-speed network environment (*e.g.*, ATM-based WANS and LANS) as well as in a highly diverse internetworking environment (*e.g.*, the current Internet).

Experience indicates that it is very difficult to specify one protocol that is optimal for all application/OS/network combinations [33]. Therefore, rather than developing a single complex all-encompassing protocol, it may be more feasible to construct an architecture that permits fine-grain selection and configuration of various protocol mechanisms [3].

ADAPTIVE also supports run-time adaptive reconfiguration in response to feedback from changes in applications requirements, transport system conditions, and network characteristics. Adaptivity is important since applications and networks are dynamically changing entities, that are not necessarily served most effectively by static solutions. Section 3.3 describes the ADAPTIVE architecture in detail.

[10]We use the term "validation" to refer to the instrumentation and measurement of transport system performance within the ADAPTIVE framework, *i.e.*, we are not investigating *verification* techniques for establishing protocol correctness [30].

[11]Other related work on transport systems typically focuses more on diverse application requirements than on network characteristics [31, 32].

(B) Reduction in Transport System Overhead: A large body of research exists on both network protocols (see [34] for a survey) and the transport systems that support them (see [35] for our survey). Techniques for reducing transport system overhead involve various combinations of the following (note, these techniques are listed roughly in increasing order of divergence from methods used in existing transport systems):

1. Utilize faster hardware for CPUs, busses, network controllers, and memory hierarchies.
2. Implement existing protocols more efficiently. For example, overall processing time may be reduced by shortening the typical instruction paths executed for each packet (*e.g.*, using a "success-oriented" protocol implementation [1]) and improving the implementation of transport system "support" services (*e.g.*, message management and demultiplexing) [13, 14, 36].
3. Design more flexible and lightweight protocols that are tailored for high-speed, low error, and low delay network environments [22, 37, 38, 39, 40, 41]. Lightweight protocols generally involve (a) a success-oriented design (*i.e.*, optimizing for the "common case" of error-free, in-sequence delivery), (b) fixed-sized, "word-aligned" headers, along with "trailer checksums," (c) fewer packet exchanges (*e.g.*, using implicit connection management and/or block acknowledgments), (d) reduction in the number of timers (*e.g.*, using sender-independent acknowledgments), and (e) reduction in unnecessary functionality (*e.g.*, making checksums optional and/or eliminating retransmissions for loss-tolerant applications).
4. Implement selected functions (*e.g.*, checksum calculations, message buffering, and demultiplexing) in special-purpose hardware [42, 43].
5. Migrate the protocol processing activities to "off-board" processors [11, 23, 44]. This helps to reduce CPU interrupts and operating system context switches.
6. Use alternative transport system architectures such as those based on (a) parallelism [1, 22, 27], (b) vertical processing architectures [10, 45], (c) flexible protocol stacks that require few layer and/or are dynamically assembled [26, 28, 31, 32], and (d) flexible transport system software that supports these flexible protocol stacks [3, 21, 38, 46].

The ADAPTIVE architecture we are developing extends prior research on flexible and adaptive transport system architectures [1, 22, 26, 28, 31, 32]. In particular, ADAPTIVE employs techniques from categories 2 (efficient implementations), 3 (flexible, lightweight designs), and 6 (alternative transport system architectures), emphasizing software that supports flexible and adaptive transport systems.[12]

(C) Focus on Both Policies and Mechanisms: Much of the the related work on transport systems focuses on providing infrastructure *mechanisms* (*e.g.*, buffering, acknowledgment schemes, retransmission timers, etc.) that determine *how* to configure and reconfigure the transport system. For example, [22] describes a variety of adaptive mechanisms that allow "on-the-fly" protocol reconfigurations. However, most existing research places less emphasis on the *policies* that determine *when* to adaptively reconfigure transport system mechanisms and *what* mechanisms the subsequent reconfigurations should contain.[13] The ADAPTIVE transport system architecture, on the other hand, is designed to specifically address both policies and mechanisms. The following are two examples of policies and mechanisms:

- The transport system may have a policy that causes the retransmission mechanism to switch from *go-back-n* to *selective repeat* in the event that (1) an application's requirements change from multicast to unicast [47] or (2) if the congestion in the network suddenly increases beyond a specified threshold

[12]To further improve its performance and versatility, the ADAPTIVE architecture may be enhanced with techniques from categories 1, 4, and 5. However, these categories are predominately hardware-oriented, and therefore are not necessarily suitable as the primary implementation strategy for a flexible and adaptive transport system architecture.

[13]Conversely, related work that addresses policies [26, 31, 32] is not specific with respect to the mechanisms that actually provide global policy enforcement. For example, adequately supporting application requirements (*e.g.*, *hard real-time delivery* and *constrained latency*) involves complicated interactions between the network and the transport systems on both local and remote hosts.

(causing packet loss due to queue overflows) [1, 2]. Note that at a later time, it may be possible to switch back, thereby reducing buffering requirements at receivers.

- The transport system may also have a policy that causes the error protection mechanism to switch from being retransmission-based to one based on forward error correction when the round-trip delay time increases beyond some threshold, (*e.g.*, when a route switches from a terrestrial link to a satellite link).

Note that in each of these cases, it is just as important to understand *when* and *what* to switch the mechanism, as it is to know how the mechanism works.

(D) Support for Controlled, Empirical Experimentation via Performance Monitoring and Measurement: It is difficult to empirically evaluate the advantages and disadvantages of different transport system designs without (1) a controlled implementation environment and (2) systematic methods for monitoring and measuring the performance [13]. Measurement is an explicit part of the ADAPTIVE methodology, which employs an iterative, feedback-driven process consisting of: (1) transport system specification and configuration, (2) experimentation, (3) analysis of the results, and (4) using feedback from (3) to refine (1).

Monitoring and measuring the performance of different transport system configurations provides feedback on the impact of selecting alternative policies and mechanisms [29]. In addition, certain metric information (*e.g.*, packet loss or round-trip delay) may be used at run-time to help determine when to reconfigure the transport system mechanisms. Metrics for monitoring and measuring transport systems include throughput, data transmission latency, jitter, packet loss, connection establishment time, and number of packet retransmissions [48, 49]. Section 3.3.3 describes these issues in greater detail.

3.2 Comparison with Alternative Solution Techniques

As mentioned above, the ADAPTIVE system is based upon a prototype-driven transport system development and experimentation methodology. The ADAPTIVE architectural design requires the following in order to support this methodology: (1) it must exhibit performance that allows realistic comparisons between alternative transport system configurations, (2) it must be flexible (to support fine-grain control over configurations), (3) it must be adaptive (to support dynamic reconfiguration), and (4) it must run in multiple host OS environments (*e.g.*, BSD and System V UNIX, the *x*-kernel, and Mach). To achieve all these goals, we considered the following four alternative approaches for implementing network protocols in transport systems [7]:

1. *General-Purpose Programming Language/OS Approach* – Many transport systems are hand-coded in a general-purpose language (*e.g.*, C or C++) using standard OS services (*e.g.*, the BSD UNIX kernel networking facilities). The primary advantage of this approach is *efficiency*, due to the close mapping from the source language to the OS and underlying hardware. The disadvantages are it is difficult to (1) verify the correctness of the solution, (2) port the software to a different OS environment, and (3) modify and extend the design and implementation.

2. *Derive from a Formal Definition Technique (FDT)* – By using an FDT such as Estelle [50], PASS [51], Petri-Nets [28], or RTAG [8], it is possible to automatically derive part of the transport system implementation based on formal specifications. The main advantages of this approach result from working at a higher-level of abstraction, which simplifies the verification and validation of protocol correctness. The primary disadvantage, however, is that these implementations may be very inefficient compared with those coded by hand. In particular, this inefficiency may bias the performance results, thereby obscuring the subtle interactions between factors being investigated (*e.g.*, FDT-generated retransmission schemes may stress the memory management facilities of the operating system differently than hand-coded versions, thus skewing packet-loss measurements).

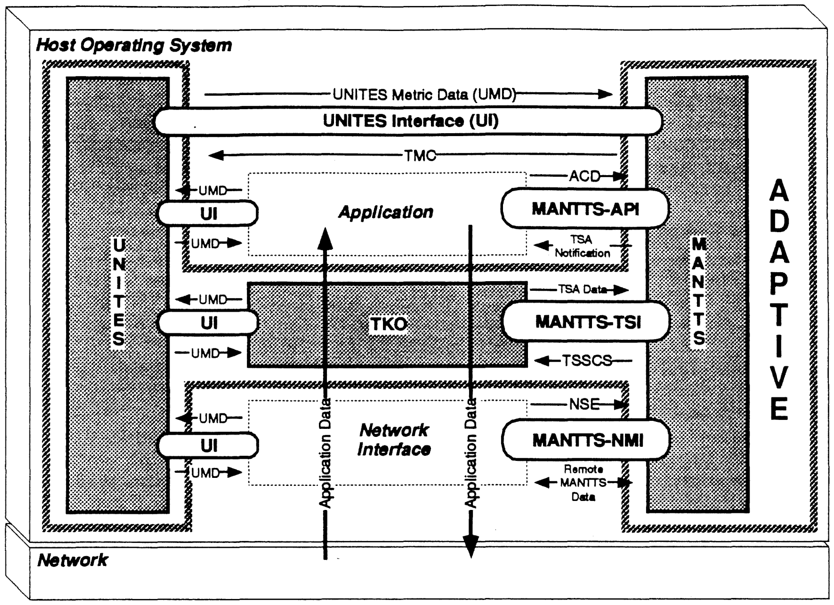

Figure 1: ADAPTIVE Architecture

3. *Use a Domain-Oriented Language-Based Approach* – This approach is more efficient (though less general) than using an FDT. It applies domain-specific knowledge about transport systems to define special-purpose languages (such as Morpheus [7]) for describing certain common protocol functions such as multiplexing and demultiplexing, addressing, error detection, and message buffering. The advantage of this approach is it enables performance optimizations that are difficult to detect with a general-purpose FDT or programming language. However, the disadvantages are that this approach does not necessarily address *adaptivity*, nor are domain-oriented programming language implementations available on all the platforms that host the ADAPTIVE system.
4. *Use a Reusable Software Component Library-Based Approach* – This approach uses a modular framework that comprises the basic building blocks of transport system functionality. For example, Avoca [3] uses the x-kernel as a run-time environment to support its micro- and virtual-protocol processing activities such as connection management, error detection, and block (re)transmission. The advantages of this approach are that it is efficient, flexible, adaptive, and fairly easy to port to different host OSs. The primary disadvantage is that it is not necessarily as efficient as the completely hand-coded or domain-oriented language-based approaches. However, we chose this approach for the ADAPTIVE system since it most closely fulfills the four goals of our system, compared with the other three methods described above.

3.3 The ADAPTIVE System Design

This section describes the architecture of the ADAPTIVE system and examines the major subsystems in detail. As shown by the shaded rectangles in Figure 1, ADAPTIVE's three main subsystems are:

1. MANTTS (*"Map Applications and Networks To Transport Systems"*) – MANTTS negotiates with peer hosts and intermediate switching nodes (ISNs) to determine policies and mechanisms that will fulfill an application's *grade of service* (GoS)[14] requirements, taking into account the dynamically changing network environment. Section 3.3.1 describes MANTTS in detail.

[14] As described in Section 3.3.1, specifying an application's grade of service requirements involves both "quality of service" (QoS) and "functionality of service" (FoS) parameters.

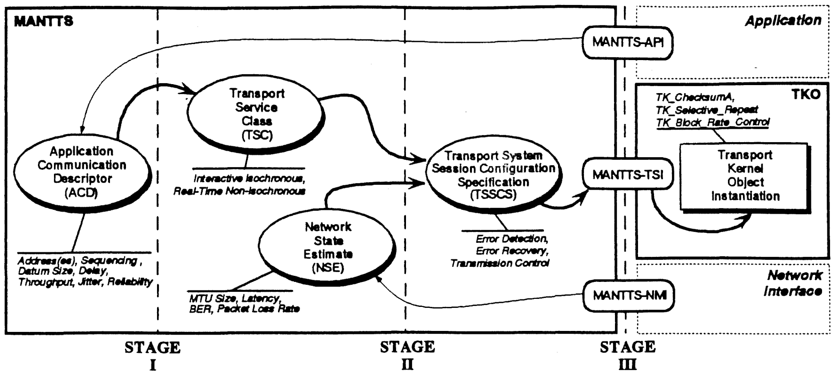
MANTTS
Application
MANTTS-API
TKO
TK_ChecksumA,
TK_Selective_Repeat
TK_Block_Rate_Control
Transport Service Class (TSC)
Application Communication Descriptor (ACD)
Interactive Isochronous, Real-Time Non-Isochronous
Transport System Session Configuration Specification (TSSCS)
MANTTS-TSI
Transport Kernel Object Instantiation
Network State Estimate (NSE)
Address(es), Sequencing, Datum Size, Delay, Throughput, Jitter, Reliability
Error Detection, Error Recovery, Transmission Control
Network Interface
MANTTS-NMI
MTU Size, Latency, BER, Packet Loss Rate
STAGE I
STAGE II
STAGE III

Figure 2: MANTTS Transformation Model

2. TKO (*"Transport Kernel Objects"*) - TKO synthesizes specially-tailored, lightweight *transport system session configurations* from a library of reusable mechanisms. Whereas MANTTS is responsible for choosing a set of policies and mechanisms, TKO is responsible for instantiating the selected mechanisms into executable session objects. Section 3.3.2 describes TKO in detail.

3. UNITES (*"UNIform Transport Evaluation Subsystem"*) - UNITES supports network traffic monitoring, metric select and collection, and performance evaluation. In addition to enabling meaningful comparisons between different transport system session configurations, UNITES provides feedback to help MANTTS and TKO determine when and how to dynamically alter certain transport system session configurations. Section 3.3.3 describes UNITES in detail.

The following sections describe each ADAPTIVE subsystem in detail. Details of the interface to the native transport/operating system are excluded for clarity.

3.3.1 Map Applications and Networks To Transport Systems (MANTTS)

MANTTS manages various resources (*e.g.*, message buffers and available communication ports) and services (*e.g.*, providing configuration and reconfiguration support) on ADAPTIVE host systems and intermediate switching nodes (ISNs). For instance, MANTTS coordinates multiple related communication sessions (*e.g.*, prioritized scheduling of synchronized multimedia streams), guides the "policy driven" transformation process that synthesizes transport system session configurations, and monitors the network to detect and respond to dynamic changes in traffic conditions. MANTTS is involved in several phases of communication: (1) during the connection negotiation phase, where it coordinates the initial configuration, (2) during the data transfer phase, where it coordinates the reconfiguration process. We discuss the activities undertaken during the configuration and reconfiguration phases below.

(A) Connection Negotiation and Configuration Phase: MANTTS interactively negotiates the requested GoS requirements when an application initiates a connection. Negotiation occurs between the local/remote application and MANTTS entities and the ISNs. This negotiation is based upon factors such as samples of the current network state and traffic volume, ISN queue lengths, and host processing loads. As shown in Figure 2, MANTTS configures a transport system session on behalf of the application via the following three-stage transformation process:

1. MANTTS transforms the GoS requirements into an appropriate set of policies known as a *Transport Service Class* (TSC) (described further below).

2. MANTTS then transforms the TSC into a *Transport System Session Configuration Specification* (TSSCS), which specifies a set of mechanisms selected to implement the TSC policies. The TSSCS is based upon information regarding static and dynamic network characteristics together with information obtained from negotiating with remote application and MANTTS entities and ISNs.
3. The mechanisms specified by the TSSCS are automatically synthesized by the *Transport Kernel Objects* (TKO) subsystem. TKO dynamically composes and instantiates specially-tailored session configurations that deliver the requested communication service to applications.

The following paragraphs describe each stage of the transformation process in greater detail.

Stage I – Transport Service Class Selection: During the first transformation stage, MANTTS selects a *transport service class* (TSC) corresponding to the application-specified communication requirements. As shown in Table 1, TSCs represent general classes of application requirements such as interactive and distributional isochronous traffic (*e.g.*, voice conversation and full-motion video), real-time traffic (*e.g.*, robotics and manufacturing control), and non-isochronous, non-real-time traffic (*e.g.*, file transfer and remote login) [6]. Each TSC embodies a set of related policy decisions selected to provide the application-requested GoS. Classifying applications by their TSC helps simplify the subsequent transport system session configuration process.[15]

The MANTTS Application Programmatic Interface (MANTTS-API) provides a service interface to the ADAPTIVE system. For example, when an application initiates a connection it passes the following parameters via the MANTTS-API:

1. *Address(es) of remote session participants* – This parameter specifies one or more addresses corresponding to the endpoints of the requested communication association (*i.e.*, if multicast is required, there will be multiple remote addresses). These addresses help to determine certain characteristics of ISNs such as MTU, propagation delay, and error rates.[16]
2. *A tuple consisting of ADAPTIVE Communication Descriptors (ACD)s* – As shown in Table 2, ACDs specify several types of information involving quality of service, functionality of service, related session synchronization groups, metric collection, and reconfiguration requests.

Stage II – Transport System Session Configuration Specification: During the second stage MANTTS transforms the selected TSC into a corresponding TSSCS.[17] The resulting TSSCS represents the set of mechanisms that enforce the policies embodied in the TSC. MANTTS produces the TSSCS by reconciling the TSC with the network characteristics associated with the indicated remote addresses. A *network state estimate* (NSE) [53] is used by the MANTTS Network Monitor Interface (MANTTS-NMI) to sample the current state of dynamic network characteristics.

The initial configuration of an ADAPTIVE transport system session (in both local and remote hosts) may chose between one of two alternatives:

1. *Explicit Negotiation* – When an application requests either explicit connection management and/or peer-to-peer negotiation, the initiating (*i.e.*, local) MANTTS entity uses an out-of-band signaling[18] channel

[15]Simplifying the configuration process is important, since the benefits from a flexible and adaptive architecture are reduced if configuration and/or reconfiguration becomes too time-consuming.

[16]Knowledge of these characteristics helps MANTTS determine the appropriate policies and mechanisms. For example, if the local application is communicating with one or more hosts on the same high-speed physical subnet the latency and error-rate may be very low compared to a wide area network. In such a situation, a specially-tailored lightweight transport system session configuration may be more effective than a general-purpose protocol such as TCP or TP4 [52].

[17]We distinguish between the TSC and TSSCS stages to emphasize the software engineering principle of "separation of concerns." This separation allows the decoupling of MANTTS's negotiation/specification-based TSC transformation policies from TKO's configuration/instantiation of the TSSCS mechanisms.

[18]Using out-of-band signaling helps to optimize the main data transfer path, since the data path is not required to interpret control packets [34].

Parameter Name	Description	Example Specifiers
Quality of Service (QoS)	Specifies the performance criteria requested by the application.	Peak and average throughput, minimum and maximum latency and jitter, error-rate probabilities, duration, etc.
Functionality of Service (FoS)	Specifies the functionality of behavior requested by the application.	Sequenced/non-sequenced delivery, (byte/packet/block)-based transmission and acknowledgment, etc.
Session Synchronization Group (SSG)	Indicates the group of sessions that are synchronized with this session.	The identifiers of related sessions.
Transport Measurement Component (TMC)	Specifies performance metrics to collect for this particular communication session.	Parameter selection, sampling rate, presentation format, etc.
Transport Service Adjustment (TSA)	Specifies the actions to perform when changes occur in the network, local or remote hosts, or ISNs.	<*condition, action*> pairs that indicate the actions to perform when certain conditions are true.

Table 2: ADAPTIVE Communication Descriptor Format

to exchange and negotiate configuration parameters[19] with remote MANTTS entities. The negotiation phase determines the GoS that the transport system is capable of providing at this point in time. Negotiation is necessary since the available GoS may be lower than that requested by the application.[20] If the negotiation process is successful, the local and remote transport system session configurations are instantiated (as described in Stage III below). On the other hand, if the negotiation/transformation is unsuccessful (*e.g.*, due to resource limitations or network partitioning) there are two alternatives: (1) refuse the connection and (2) re-negotiate at a lower grade of service.

2. *Implicit Negotiation* – Latency-sensitive applications (*e.g.*, transaction-oriented applications like network file servers) may not be able to incur any GoS negotiation delay. In such cases, MANTTS supports several types of fast connection establishment:
   - implicit connection management may be used, with configuration information piggybacked along with the application's first PDU. To support implicit negotiation, remote MANTTS entities must supply reasonable values for default configurations.
   - Another method for reducing latency is to allow applications to send small quantities of data over the signaling channel (this is similar to the VMTP *Message Control Block* feature, which allows small amounts of data to be transmitted inside the packet header [24]).

Figure 3 illustrates the separate control and data paths used by MANTTS signaling and data units during the negotiation process and subsequent data transfer. For long-term, high-bandwidth connections the additional time spent negotiating GoS does not significantly impact the overall performance. In fact, overall performance should improve, since the configurations more accurately reflect the application requirements and network characteristics [34]. In addition, the negotiation process may be combined with explicit connection management (*e.g.*, during the initial 3-way handshake). Finally, the ISNs may also become involved in the negotiation process, if resource reservation is required [55].

[19]Negotiated parameters include buffer space, initial window advertisements and scaling factors [54], segment size and maximum transmission unit (MTU) sizes, priorities for message delivery and scheduling, different mechanisms for error protection and transmission control, timer settings for delayed acknowledgments and retransmissions, etc.

[20]Note that negotiation need not determine an *optimal* configuration, as long as it produces one that meets application requirements (which may have ranges of tolerance, for instance).

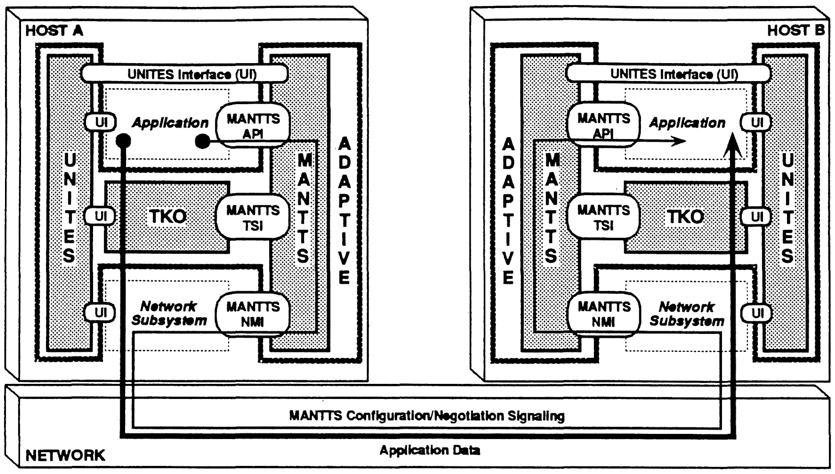

Figure 3: Connection Configuration

Stage III - Transport System Synthesis: In the third stage, MANTTS submits the TSSCS to the *Transport Kernel Object* (TKO) subsystem via the MANTTS Transport System Interface (MANTTS-TSI). TKO transforms the TSSCS into a *transport system session configuration* that is specially-tailored for the particular applications and networks involved. This final transformation stage involves (1) synthesizing the specified mechanisms (*e.g.*, transmission control and error protection) from the TKO class object library, (2) instantiating a protocol interpreter that invokes operations on these objects in a particular order, and (3) instrumenting the synthesized mechanisms using the UNITES metric collection and reporting facilities to satisfy the application's metric collection requests.[21] Section 3.3.2 describes the TKO subsystem in greater detail.

(B) Data Transfer and Reconfiguration Phase: In order to adapt to dynamic changes in application requirements and network characteristics, the ADAPTIVE architecture also supports the reconfiguration of transport system sessions. Reconfigurations may be triggered either explicitly or implicitly. Explicit reconfiguration is initiated by a local or remote application request (*e.g.*, changing from unicast to multicast communication). If a new TSSCS is generated to satisfy the request, the remote MANTTS entities are notified, and the corresponding transport system session configurations are updated. Explicit reconfiguration requests are sent via the same out-of-band signaling channel used for connection negotiation.

MANTTS automatically detects situations where implicit reconfiguration is necessary. These situations are typically initiated from either (1) requests by a local or remote MANTTS (*e.g.*, in response to increase or decrease in buffer space or processor load) or (2) from a detected change in network conditions (*e.g.*, increased congestion, ISN failure, or different route[22]). The reconfiguration process is similar to the initial configuration process, and may involve decreasing or increasing the GoS. It is non-trivial to adaptively reconfigure a transport system once a connection is established. For example, it is essential that distributed session context is properly synchronized to insure error-free and hazard-free delivery.

During reconfiguration, one of the following three actions occurs:

1. *Adjust the TSSCS* - Replace or adjust one or more mechanisms used by the transport system to compensate for the indicated change (*e.g.*, increase inter-PDU gap used by rate control mechanism due to perceived network congestion). No change occurs in the Transport Service Class.

[21]Note, this involves metric-collection instrumentation based on both application TMC requests, along with global UNITES data collection requests (described in Section 3.3.3).

[22]For example, if an active connection starts using a longer-delay path (*e.g.*, if it switches from a terrestrial link to a satellite link) it may be necessary to reconfigure mechanisms (*e.g.*, retransmission scheme or window scaling options.

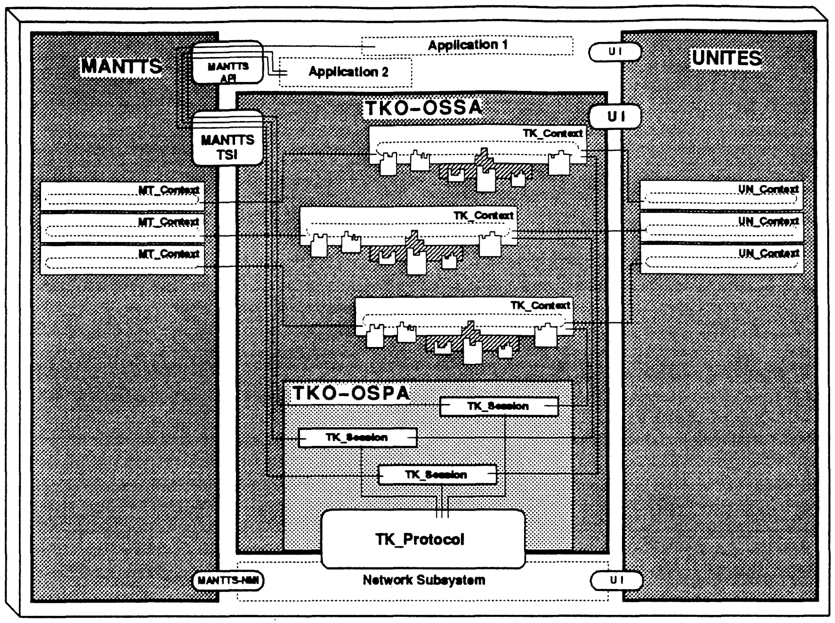

Figure 4: TKO OS Protocol and Session Architectures

2. *Adjust the TSC* - Change the Transport Service Class to provide a GoS that is better suited to the new conditions (*e.g.*, benefitting an application that has changed video coding schemes and now requires isochronous service). This change potentially generates a completely new or partially modified TSSCS.
3. *Application-specific* - Application is notified of changes in operating conditions (*e.g.*, a reduction in receiver's buffer space) via a *call-back* mechanism, allowing it to react appropriately (*e.g.*, begin transmitting data using an application-specific compression scheme).

3.3.2 Transport Kernel Objects (TKO)

The Transport Kernel Objects (TKO) subsystem provides a modular, flexible, and extensible framework for configuring and reconfiguring transport systems. As shown in Figure 4, the TKO subsystem consists of two main components that are described below: (1) TKO OS Protocol Architecture (TKO-OSPA) and (2) TKO OS Session Architecture (TKO-OSSA). Each component performs a well-defined set of services for the transport system.

(A) TKO OS Protocol Architecture (TKO-OSPA): TKO-OSPA is a library of C++ classes that insulate the transport system from the underlying operating system environment.[23] These classes provide a uniform interface for accessing protocol support services such as timers, message buffering, and protocol graph operations (*e.g.*, *inserting*, *deleting*, and/or *altering* protocol objects). There are four fundamental TKO-OSPA classes: `TK_Event`, `TK_Message`, `TK_Protocol` and `TK_Session`:

1. `TK_Event` - Many protocols must respond to temporal events such as retransmission timer expiration or periodic update requests [57]. The `TK_Event` class defines an infrastructure for event management, with operations like `TK_Event::schedule`, `TK_Event::happen`, and `TK_Event::cancel`.

[23]The TKO-OSPA design is influenced by the *x*-kernel [10] and Conduit [56] communication systems. ADAPTIVE is presently being hosted on both the *x*-kernel and System V release 4 operating systems.

`TK_Event` objects schedule themselves to happen one or more times (*i.e.*, they are one-shot or periodic), they may be cancelled, and they are triggered to happen asynchronously by the operating system's timer facility.

2. `TK_Message` – Empirical studies indicate that memory-to-memory copying is a significant source of transport system overhead [58]. Therefore, some form of buffer management is necessary to avoid unnecessary copying when (1) moving messages between protocol layers and (2) when adding or deleting headers and trailers [59]. The `TK_Message` class provides a uniform interface for services that create, copy, prepend, and split messages. `TK_Message` objects are logically divided into two distinct regions: the *header* and the *data*. The data region efficiently supports "lazy copying" operations and fragmenting and assembling of data chunks. Likewise, the header region allows operations (*e.g.*, `TK_Message::push` and `TK_Message::pop`) that efficiently prepend header information onto a message and later strip it off.
3. `TK_Protocol` – Protocol suites are typically specified as a hierarchy of layers, where each layer performs a well-defined set of services for layers above it, and makes use of services from layers below it [60]. The `TK_Protocol` class provides uniform interfaces for protocol objects that compose protocol suites like TCP/IP, OSI, and XNS. The service interface for a `TK_Protocol` object includes (1) management operations that compose protocol graphs representing the relationships between various protocol entities and (2) operations that create, destroy, and pass messages to `TK_Session` objects. Each `TK_Protocol` object contains 0 or more `TK_Session` objects.
4. `TK_Session` – As shown in Figure 4, the `TK_Session` class forms the junction between the TKO-OSPA and TKO-OSSA services (described in the following section). `TK_Session` objects encapsulate certain connection context information (*e.g.*, local and remote addresses) that are needed to process incoming and outgoing messages. Associated with this context is a set of operations used for (1) sending and receiving `TK_Message` objects that flow through `TK_Session` objects, (2) dynamic session attachment (*i.e.*, allocating and linking together new sessions to form session graphs), and (3) dispatching system calls that store and/or retrieve session control information (*e.g.*, determining host and peer network addresses or determining the MTU for a given network interface, etc.).

These four fundamental classes, along with several auxiliary classes, provide the infrastructure to support TKO-OSSA configurations described below.

(B) TKO OS Session Architecture (TKO-OSSA): As with the TKO-OSPA, the TKO-OSSA also contains a library of reusable C++ transport system mechanisms. However, each class in the TKO-OSSA library encapsulates "finer-grain" mechanisms associated with session processing activities (*e.g.*, connection management, error protection, and data transmission control). The following paragraphs describe the organization of components in the TKO-OSSA, explains how the components are synthesized and combined, and discusses several standard performance optimization techniques.

Base Classes and Derived Subclasses: To support flexible configuration and adaptive reconfiguration, TKO-OSSA is organized as a set of C++ inheritance hierarchies.[24] These hierarchies are "rooted" at abstract base classes (ABCs) that provide uniform interfaces for common protocol processing activities. To synthesize the appropriate protocol functionality that flexibly supports application-specific requirements involves (1) *deriving* (*i.e.*, inheriting the context and operations) specialized subclasses from the ABCs and (2) instantiating objects of these subclasses. As shown in Figure 5, TKO-OSSA provides a standard set of abstract base classes that correspond to the basic mechanisms used in protocol processing (*e.g.*, `Connection_Management`, `Synchronization_Management`, `Reliability_Management`, and `Transmission_Management`).

[24]The TKO library is implemented in C++ since (1) it directly supports dynamism and specialization via dynamic binding and inheritance and (2) it integrates context information with operations to support object-oriented design and programming.

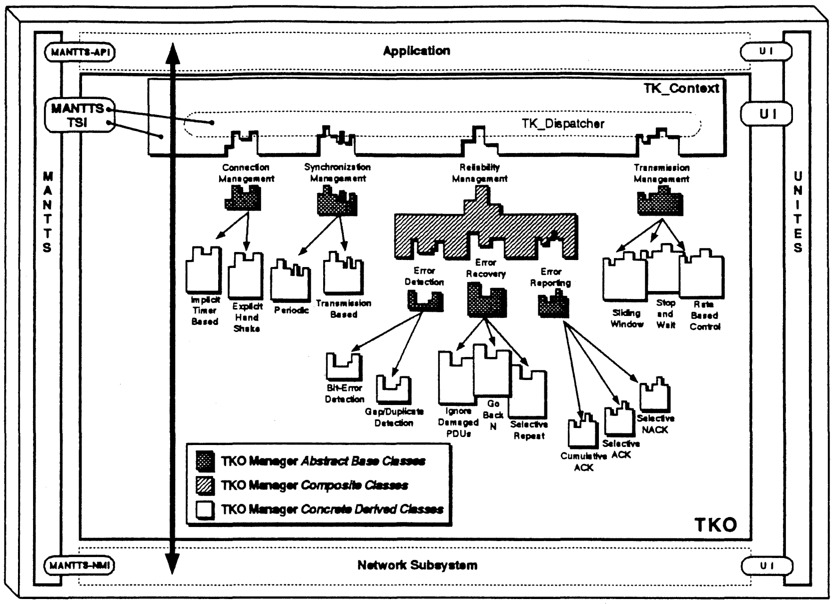

Figure 5: TKO Context

Particular instances of these mechanisms are derived from the appropriate base class (*e.g.*, `Sliding_Window` is derived from `Transmission_Management`). The `Reliability_Management` class shown in Figure 5 is an example of a *Composite Component.* Composite components are used to manage objects of several related subcomponents that interoperate in well-defined ways (*e.g.*, the detection, recovery and reporting mechanisms of a `Reliability_Management` mechanism). Mechanisms implemented using composite components allow the replacement of one or more submechanisms, facilitating rapid prototyping and incremental protocol development.

As illustrated in Figure 4, each open `TK_Session` object points to a specially-tailored table containing session context information (`TK_Context`). Figure 5 enlarges the view of the `TK_Context`, which contains a collection of pointers to abstract base classes that together comprise the protocol state machine.[25] Each entry in the `TK_Context` points to a derived subclass that (1) is specialized to perform the appropriate protocol processing function, (2) maintains the necessary context information (*e.g.*, round-trip time estimates, sequence numbers, or flow control window advertisements) to support the function, and (3) provides the means for replacing itself with a different derived class (*i.e.*, to support reconfiguration by dynamically changing the referenced objects). This encapsulation of context and function allows protocol mechanisms to be replaced and recombined without incurring time-consuming overhead.

TK_Dispatcher: The `TK_Dispatcher` is responsible for overseeing the configuration and reconfiguration of `TK_Context` objects. It receives the TSSCS from the MANTTS-TSI and transforms it into an efficient, economical instantiation. `TK_Dispatcher` is responsible for monitoring the transport system conditions specified in the Transport System Adjustment (TSA), and for coordinating reconfiguration of the transport system by instantiating the correct object(s) and initiating the replacement(s) via the `segue` mechanism built into all abstract base classes.

[25]The entries in this table may be selected at either configuration-time or run-time. This differs from the BSD 4.3 UNIX approach, which statically binds protocol operations to TCBs through pointers to C functions, *i.e.*, all sessions associated with a particular protocol object use the same bindings. TKO-OSSA, on the other hand, supports fine-grain, per-session control over these bindings.

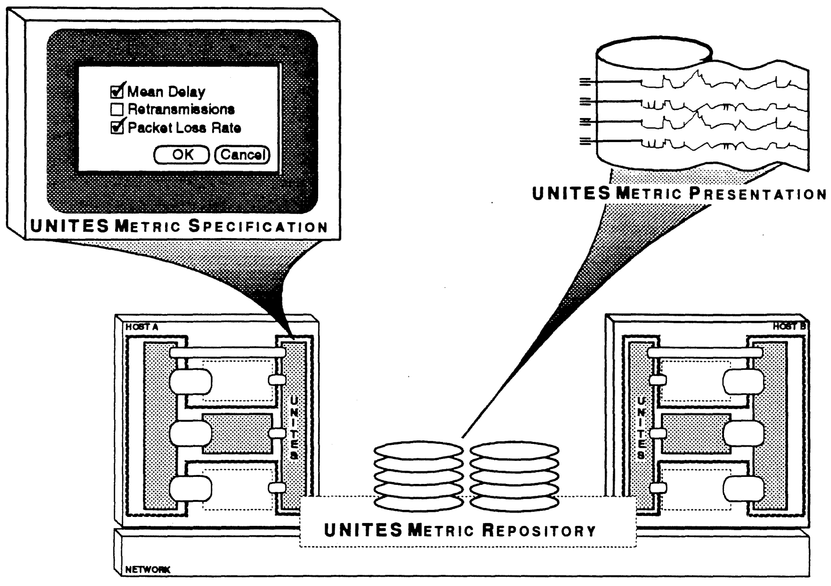

Figure 6: UNITES Architecture

Performance Optimizations: By default, the TKO-OSSA dynamism is supported by the C++ dynamic binding mechanism (using C++'s "virtual functions" [61]). Although dynamic binding enhance *flexibility* (and thereby facilitates *prototyping* and *experimentation*), it increases processing overhead somewhat due to the extra level of indirection required to dispatch the C++ virtual functions. To reduce this overhead, TKO employs a technique known as *customization* [62], which instantiates non-dynamically bound configurations for circumstances where performance is preferred over flexibility.

To further optimize the instantiation process, the TKO-OSSA maintains a cache of customized "`TK_Templates`" that provide default transport system session configurations for commonly requested TSSCSs. These "fast path" templates reduce the complexity and the duration of the option negotiation phase. There are two general types of `TK_Templates`:

1. *Static – i.e.*, guaranteed not to change. This allows more efficient implementation, since the mechanisms can be completely customized (*e.g.*, inline expanded rather than dynamically dispatched).
2. *Reconfigurable – i.e.*, may change at some point during the communication session. This is generally more flexible, but less efficient, since there is some additional indirection required (though not as much as is needed by a dynamically synthesized configuration).

If a pre-assembled `TK_template` does not exist to match a TSSCS request, TKO-OSSA dynamically synthesizes one [63].

3.3.3 Uniform Transport and Evaluation Subsystem (UNITES)

A fundamental goal of the ADAPTIVE system is to provide a framework that supports controlled hypothesis testing of different transport system session configurations. For example, we are currently experimenting with alternative protocol designs to determine configurations that best serve a particular set of multimedia application requirements and underlying network characteristics. The UNITES subsystem facilitates experimentation by coordinating *metric specification*, *metric collection*, *metric analysis*, and *metric presentation*. Information obtained from UNITES metrics quantifies trade-offs and interactions among different configurations, allowing meaningful design and implementation evaluations.

UNITES metrics are divided into two classes: *blackbox* and *whitebox*, that differ depending on whether or not the metric collection mechanisms require internal instrumentation of transport system configurations. Blackbox metrics are collected without knowledge of implementation details. They include throughput (defined as the number of packets transmitted per second), latency (defined as response time for interactive traffic), jitter, and packet loss (defined as the number of "successful" packets[26]). Whitebox metrics, on the other hand, require internal instrumentation of the configurations. Whitebox metrics include latency of connection establishment and termination, number of packet (re)transmissions, the number of instructions required to execute a protocol function, and interrupt and scheduling overhead. Both blackbox and whitebox metrics assist the tuning of transport system configurations by indicating the performance consequences of certain design and implementations decisions.

As illustrated in Figure 6, the UNITES Metric Repository stores the metric information in a database.[27] Users can access this information in various ways such as interactive graphic displays or standard network management protocols such as SNMP or CMIP. This metric data is presented in either a systemwide, per-host, or per-connection manner. There are two primary ways that UNITES is used to instrument transport system sessions:

1. Application programs indicate metrics they want UNITES to collect via the Transport Measurement Component (TMC) parameter in the ADAPTIVE Communication Descriptor (ACD). The TKO subsystem then selectively instruments the synthesized configurations and the metrics are automatically collected during run-time.
2. To support experimentation, metrics also may be requested using either a graphics-based or language-based interface. Sjodin *et al.*'s work is an example of a language-based approach. They define a specification language that indicates what measurements to collect and what traffic to generate [48]. Figure 6 shows a graphical interface for specifying certain metrics to collect on a per-host basis.

4 Summary

ADAPTIVE is an integrated framework that exhibits: (1) support for application and network diversity and dynamism via a *flexible* and *adaptive* architecture, (2) reduction in transport system overhead, (3) focus on policies and mechanisms, and (4) feedback-guided monitoring and measurement. A flexible software architecture like ADAPTIVE is essential for supporting an "experimentation-based" methodology. ADAPTIVE facilitates precise measurement of the application and network performance that results from selectively changing certain transport system mechanisms (*e.g.*, measuring the effect of switching from *implicit* to *explicit* connection management or from *selective repeat* to *go-back-n* retransmission).

We are currently designing and implementing a prototype implementation written in C++ that runs on both the x-kernel and System V STREAMS. We plan to use this prototype to experiment with different transport system configurations that support multimedia applications (*e.g.*, network voice and video) running on several different networks (*e.g.*, Ethernet, Tree Network [64], DQDB, and FDDI).

Acknowledgments

We would like to thank Hung Huang, Unmesh Rathi, and Girish Kotmire for their help in designing the ADAPTIVE system and for proof-reading earlier drafts of this paper.

[26]Where success is operationalized as (100 * (1 - (# lost + # dropped + #duplicated + #sent))).

[27]As noted in [29], when many active connections are instrumented there may be too much data to collect and process in real-time.

References

[1] Z. Haas, "A Protocol Structure for High-Speed Communication Over Broadband ISDN," *IEEE Network Magazine*, pp. 64–70, January 1991.

[2] T. F. L. Porta and M. Schwartz, "Architectures, Features, and Implementation of High-Speed Transport Protocols," *IEEE Network Magazine*, pp. 14–22, May 1991.

[3] S. W. O'Malley and L. L. Peterson, "A Dynamic Network Architecture," tech. rep., Department of Computer Science, University of Arizona, Tucson, Ariz., October 1991.

[4] H. E. Meleis and D. N. Serpanos, "Memory Management in High-Speed Communication Subsystems," in *Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Feb. 1992.

[5] M. Zitterbart, "High-Speed Protocol Implementations Based on a Multiprocessor-Architecture," in *Proceedings of the 1st International Workshop on High-Speed Networks*, pp. 151–163, May 1989.

[6] M. Zitterbart and B. Stiller, "A Concept for A Flexible High Performance Transport System," in *Telecommunications and Multimedia Applications in Computer Science*, pp. 365–374, Oct. 1991.

[7] M. B. Abbott and L. L. Peterson, "A Language-Based Approach to Protocol Implementation," Tech. Rep. 92-2, University of Arizona, 1992.

[8] D. P. Anderson, "Automated Protocol Implementation with RTAG," *IEEE Transactions on Software Engineering*, vol. 14, pp. 291–300, Mar. 1988.

[9] G. Blair, G. Coulson, F. Garcia, D. Hutchinson, and D. Shepherd, "Towards New Transport Services to Support Distributed Multimedia Applications," in *4th IEEE ComSoc International Workshop on Multimedia Communications*, (Monterey, California), pp. 250–259, IEEE, 1992.

[10] N. C. Hutchinson and L. L. Peterson, "The *x*-kernel: An Architecture for Implementing Network Protocols," *IEEE Transactions on Software Engineering*, vol. 17, pp. 64–76, January 1991.

[11] H. Kanakia and D. R. Cheriton, "The VMP Network Adapter Board (NAB): High-Performance Network Communication for Multiprocessors," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Stanford, CA), pp. 175–187, ACM, Aug. 1988.

[12] P. Steenkiste, "Analysis of the Nectar Communication Processor," in *Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Feb. 1992.

[13] R. W. Watson and S. A. Mamrak, "Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices," *ACM Transactions on Computer Systems*, vol. 5, pp. 97–120, May 1987.

[14] D. D. Clark, "Modularity and Efficiency in Protocol Implementation," *Network Information Center RFC 817*, pp. 1–26, July 1982.

[15] S. J. Leffler, M. McKusick, M. Karels, and J. Quarterman, *The Design and Implementation of the 4.3BSD UNIX Operating System*. Addison-Wesley, 1989.

[16] D. R. Cheriton, "The V Distributed System," *Communications of the ACM*, vol. 31, March 1988.

[17] R. V. Renesse, H. V. Staveren, and A. S. Tanenbaum, "Performance of the Amoeba Distributed Operating System," *Software - Practice and Experience*, vol. 19, pp. 223–234, March 1989.

[18] M. N. Group, "Network Server Design," Tech. Rep. CMS-CS-89-31, Carnegie Mellon University, August 1989.

[19] B. B. Welch, "The Sprite Remote Procedure Call System," Tech. Rep. UCB/CSD 86/302, Computer Science Division (EECS), University of California, Berkeley, CA 94720, June 1986.

[20] UNIX Software Operations, *UNIX System V Release 4 Programmer's Guide: STREAMS*. Prentice Hall, 1990.

[21] J. M. Zweig, "The Conduit: a Communication Abstraction in C++," in *USENIX C++ Conference Proceedings*, pp. 191–203, USENIX Association, April 1990.

[22] T. F. L. Porta and M. Schwartz, "Design, Verification, and Analysis of a High Speed Protocol Parallel Implementation Architecture," in *Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Feb. 1992.

[23] G. Chesson, "XTP/PE Design Considerations," in *Proceedings of the 1st International Workshop on High-Speed Networks*, May 1989.

[24] D. R. Cheriton, "VMTP: Versatile Message Transaction Protocol Specification," *Network Information Center RFC 1045*, pp. 1–123, Feb. 1988.

[25] D. F. Box, D. C. Schmidt, and T. Suda, "Alternative Approaches to ATM/Internet Interoperation," in *IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, pp. 1–5, IEEE, 1992.

[26] T. Plagemann, B. Plattner, M. Vogt, and T. Walter, "A Model for Dynamic Configuration of Light-Weight Protocols," in *IEEE Third Workshop on Future Trends of Distributed Systems*, pp. 1–9, IEEE, 1992.

[27] M. Zitterbart, "High-Speed Transport Components," *IEEE Network Magazine*, pp. 54–63, January 1991.

[28] B. Heinrichs, "Versatile Protocol Processing for Multimedia Communications," in *4th IEEE ComSoc International Workshop on Multimedia Communications*, (Monterey, California), pp. 160–169, IEEE, 1992.

[29] R. C. Albert Li, A. Fawaz, S. Sachs, P. Varaiya, and J. Walrand, "The Programmable Network Prototyping System," *IEEE Computer*, vol. 22, pp. 67–76, May 1989.

[30] G. J. Holzmann, *Design and Validation of Computer Protocols*. Englewood Cliffs, NJ: Prentice Hall, 1991.

[31] B. Stiller, "PROCOM: A Manager for an Efficient Transport System," in *Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Feb. 1992.

[32] C. Tschudin, "Flexible Protocol Stacks," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Zurich Switzerland), pp. 197–205, ACM, Sept. 1991.

[33] L. Peterson and S. O'Malley, "TCP Extensions Considered Harmful," *Network Information Center RFC 1263*, pp. 1–19, 1991.

[34] W. A. Doeringer, D. Dykeman, M. Kaiserswerth, B. W. Meister, H. Rudin, and R. Williamson, "A Survey of Light-Weight Transport Protocols for High-Speed Networks," *IEEE Transactions on Communication*, vol. 38, pp. 2025-2039, November 1990.

[35] D. C. Schmidt and T. Suda, "Exploring the Computing Dimensions of Communications Software," Tech. Rep. 92-26, University of California, Irvine, February 1992.

[36] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An Analysis of TCP Processing Overhead," *IEEE Communications Magazine*, vol. 27, pp. 23–29, June 1989.

[37] D. D. Clark, M. L. Lambert, and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol," *Network Information Center RFC 998*, pp. 1–21, Mar. 1987.

[38] A. Fraser, "The Universal Receiver Protocol," in *Proceedings of the 1st International Workshop on High-Speed Networks*, pp. 19–25, May 1989.

[39] R. W. Watson, "The Delta-*t* Transport Protocol: Features and Experience," in *Proceedings of the 1st International Workshop on High-Speed Networks*, May 1989.

[40] D. R. Cheriton and C. L. Williamson, "VMTP as the Transport Layer for High-Performance Distributed Systems," *IEEE Communications Magazine*, vol. 27, pp. 37–44, June 1989.

[41] Protocol Engines Incorporated, Santa Barbara, *XTP Protocol Definition*, September 1990.

[42] C. M. Woodside and J. R. Montealegre, "The Effect of Buffering Strategies on Protocol Execution Performance," *IEEE Transactions on Communications*, vol. 37, pp. 545–554, June 1989.

[43] M. L. Bailey, M. A. Pagels, and L. L. Peterson, "The *x*-chip: An Experiment in Hardware Demultiplexing," in *Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Feb. 1992.

[44] E. C. Cooper, P. A. Steenkiste, R. D. Sansom, and B. D. Zill, "Protocol Implementation on the Nectar Communication Processor," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Philadelphia, PA), pp. 135–144, ACM, Sept. 1990.

[45] J. Jain, M. Schwartz, and T. Bashkow, "Transport Protocol Processing at GBPS Rates," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Philadelphia, PA), pp. 188–199, ACM, Sept. 1990.

[46] D. D. Clark, "The Structuring of Systems Using Upcalls," in *Proceedings of the Tenth Symposium on Operating Systems Principles*, (Shark Is., WA), 1985.

[47] R. M. Sanders and A. C. Weaver, "The Xpress Transfer Protocol (XTP) — A Tutorial," *ACM Computer Communication Review*, vol. 20, pp. 67–80, Oct. 1990.

[48] P. Gunningberg, M. Bjorkman, E. Nordmark, S. Pink, P. Sjodin, and J.-E. Stromquist, "Application Protocols and Performance Benchmarks," *IEEE Communications Magazine*, vol. 27, pp. 30–36, June 1989.

[49] P. Sjodin, P. Gunningberg, E. Nordmark, and S. Pink, "Towards Protocol Benchmarks," in *Proceedings of the 1st International Workshop on High-Speed Networks*, May 1989.

[50] S. T. Vuong, A. C. Lau, and R. I. Chan, "Semiautomatic Implementation of Protocols Using an Estelle-C Compiler," *IEEE Transactions on Software Engineering*, vol. 14, pp. 384–393, Mar. 1988.

[51] M. Farber and T. Schutt, "High-Speed Protocol Implementations: An Experience with PASS," in *Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Feb. 1992.

[52] M. S. Atkins, S. T. Chanson, and J. B. Robinson, "LNTP - An Efficient Transport Protocol for Local Area Networks," in *Proceedings of the Conference on Global Communications (GLOBECOM)*, pp. 705–710, 1988.

[53] D. C. Feldmeier and E. W. Biersack, "Comparison of Error Control Protocols for High Bandwidth-Delay Product Networks," in *Protocols for High-Speed Networks II*, IFIP, 1991. a.

[54] V. Jacobson and R. Braden, "TCP Extensions for Long-Delay Paths," *Network Information Center RFC 1072*, pp. 1–16, Oct. 1988.

[55] D. P. Anderson, R. G. Herrtwich, and C. Schaefer, "SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet," Tech. Rep. 90-006, International Computer Science Institute, February 1990.

[56] J. M. Zweig, "An Object-Oriented Framework for Implementing Network Protocols," Master's thesis, University of Illinois at Urbana-Champaign, 1991.

[57] A. N. Netravali, W. D. Roome, and K. Sabnani, "Design and Implementation of a High Speed Transport Protocol," *IEEE Transactions on Communications*, 1990.

[58] D. D. Clark and D. L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Philadelphia, PA), pp. 200–208, ACM, Sept. 1990.

[59] S. W. O'Malley, M. B. Abbott, N. C. Hutchinson, and L. L. Peterson, "A Transparent Blast Facility," *Journal of Internetworking*, vol. 1, December 1990.

[60] A. S. Tanenbaum, *Computer Networks (Second Edition)*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[61] Bjarne Stroustrup, *The Annotated C++ Reference Manual*. Addison-Wesley, 1990.

[62] C. Chambers and D. Ungar, "Customization: optimizing compiler technology for SELF (a dynamically-typed object-oriented language)," in *Proc. SIGPLAN*, ACM, 1989.

[63] C. Pu, H. Massalin, and J. Ioannidis, "The Synthesis kernel," *Computing Systems*, vol. 1, pp. 11–32, Winter 1988.

[64] H. K. Huang, T. Suda, and Y. Noguchi, "LAN With Collision Avoidance: Switch Implementation and Simulation Study," in *Proceedings of the 15th Conference on Local Computer Networks*, 1990.