TU

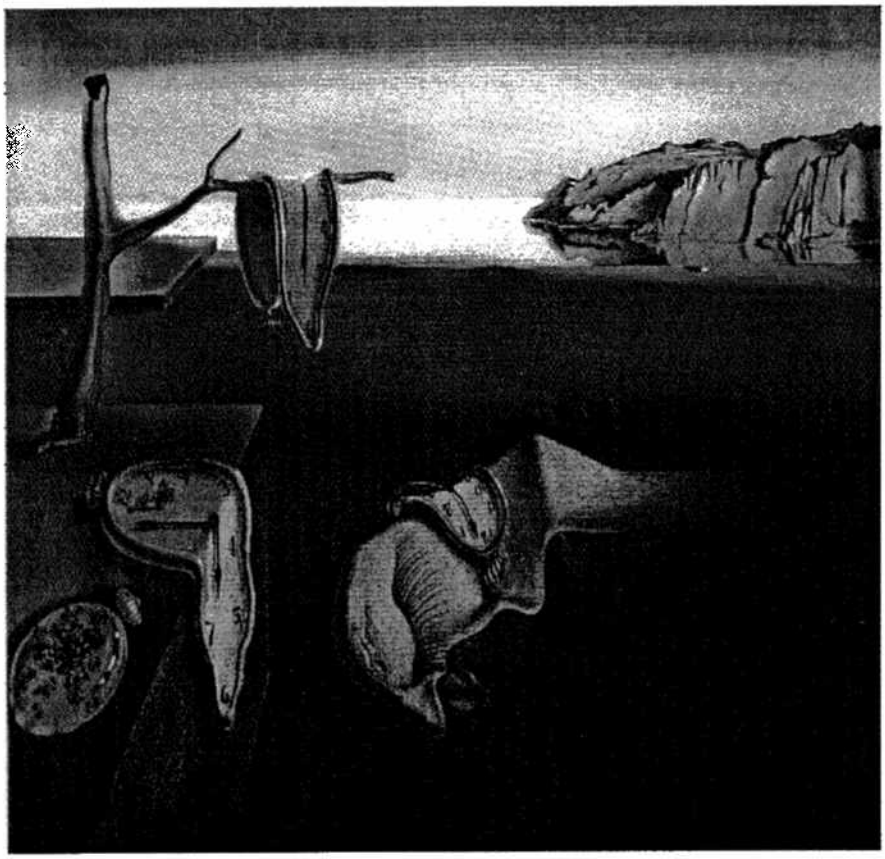
Institut für Automation
Abt. für Automatisierungssysteme

Technische
Universität
Wien

Projektbericht Nr. 183/1-58
August 1995

Project Proposal SSCMP: Sequenced Synchronized Clock Multicast Protocol

Ulrich Schmid, Dietmar Loy and Wolfgang Kastner



Salvador Dali, "Die Beständigkeit der Erinnerung"

Project Proposal SSCMP
Sequenced Synchronized Clock Multicast Protocol

ULRICH SCHMID
Technical University of Vienna
Department of Automation 183/1
Treitlstraße 3, A-1040 Wien
Email: s@auto.tuwien.ac.at

DIETMAR LOY
Technical University of Vienna
Department of Computer Technology 384
Gußhausstraße 27, A-1040 Wien
Email: loy@ict.tuwien.ac.at

WOLFGANG KASTNER
Technical University of Vienna
Department of Automation 183/1
Treitlstraße 3, A-1040 Wien
Email: k@auto.tuwien.ac.at

July 1995

Abstract

This proposal describes a project devoted to development and analysis of a novel reliable data transmission protocol, called the *Sequenced Synchronized Clock Multicast Protocol* (SSCMP), providing timely delivery of messages and atomic broadcasting. Our protocol differs from existing ones by the fundamental role dedicated to *time* in all the algorithms employed. In particular, contrasting usual approaches based on pure sequence numbers, SSCMP provides sequenced at-most-once delivery of messages by means of the new timer-based connection management protocol *Sequenced Synchronized Clock Message*[1] *Protocol* relying on messages incorporating timestamps; a first description and analysis of the latter has already been accepted by *Computer Networks and ISDN Systems* ([SP95]). Since timestamped messages are required for timely multiplexing/transmission scheduling algorithms and synchronous atomic broadcasting (and usually needed at the application level anyway), SSCMP offers conceptual coherence and improved performance at the same time.

Keywords: computer communications, distributed real-time systems, timer-based connection management protocols, sequenced at-most-once message delivery, real-time communication scheduling, multi-access channels, atomic broadcast, synchronized clocks.

[1] We just changed the meaning of the letter 'M' in the connection management protocol SSCMP of [SP95] from *message* to *multicast* to arrive at the SSCMP of our proposal.

Contents

1 Introduction

This project proposal originates in the problem of providing a high-performance distributed real-time system required for building the research prototype of our monitoring system *Versatile Timing Analyzer* VTA[2]. To save development cost and time in the VTA project, it was quite natural to rely on standard hardware and software technology where possible, e.g. VME-CPUs running the pSOS^{+m} multiprocessor operating system kernel, and to add lacking features: (1) high-accuracy synchronized clocks, and (2) a high-performance, real-time communication subsystem for Ethernet.

To provide synchronized clocks, the authors initiated their research project SynUTC[3] aiming at the development of hardware and software implementing high-accuracy (μs-range) local clocks by means of incorporating GPS receivers, see [Sch95] for related issues. We may say that this project is getting on very well, both w.r.t. to theory and practice, see Section 3.2 for a brief report on the current status.

When exploring possible solutions for the real-time communication subsystem required, we became more and more unsatisfied with the incoherency that resulted from putting together things like Le Lann's *Deterministic Ethernet* (cf. [LeL87]), a 802.x data link layer protocol, and Cristian's atomic broadcast protocol (cf. [Cri90]), for example. Therefore, we started looking for an approach that allows to deal with such diverse issues as timely delivery, reliable data transmission, and atomic broadcasting in a coherent way.

The unifying entity, *timestamped messages*, eventually materialized when we learned about the (unsequenced) *Synchronized Clock Message Protocol* (SCMP) of [LSW91]. We soon realized that this idea was the missing link for devising a coherent solution to our original problem, and a promising direction of research as well. In fact, in the course of setting up the proposal, we discovered and analyzed the core of a novel timer-based connection management protocol called *Sequenced Synchronized Clock Message Protocol* (SSCMP), which has a number of advantages over existing connection management protocols. Most notably, it provides sequenced at-most-once delivery even in the case of clock synchronization failures and is strikingly simple, see Section 2.1 for more details. The actual merits and novelty of our approach should be judged by the fact that a paper on SSCMP was readily accepted for publication in the well-respected international journal *Computer Networks and ISDN Systems*. Even more important, *Gerard Le Lann* from INRIA Rocquencourt has expressed his interest in establishing a cooperation on the SSCMP project, which will certainly have major impacts on the quality of our work.

Of course, as pointed out by the referees of [SP95], much work —i.e., the project in question— remains to be done to convert our core idea to a fully engineered and analyzed protocol. More precisely, it is the purpose of this project to develop software and hardware for a fully-engineered and proven-correct atomic broadcast protocol suitable for high-performance, fault-tolerant real-time systems built on top of existing technology. Needless to say, the results of the abovementioned SynUTC-project are an ideal basis for the SSCMP-project, and it is even possible to exploit certain synergies between both projects. Note that we deliberately delayed submitting this proposal up to now, in order

[2]Supported by the Austrian Science Foundation, grant no. P8390-TEC.
[3]Supported by the Austrian Science Foundation, grant no. P10244-ÖMA.

to be certain that the results of SynUTC will be available when we actually need them in SSCMP.

The combined results of SSCMP and SynUTC should be immediately applicable in practice, both for implementing our VTA and, most importantly, for industrial real-time applications (where similar technology is often employed[4]).

However, note carefully, that targeting this project to fault-tolerant real-time applications is dictated by our desire to limit the amount of work to be done within a single project. Past experience has shown that reasonable self-denial contributes much to a projects success. Nevertheless, much more research could (and should) be performed on SSCMP. For example, we are planning a future project devoted to extending SSCMP by guaranteed QoS (*Quality of Service*) on top of very-high speed networks as required for multimedia applications.

In fact, we are reasonably convinced that the ideas underlying SSCMP are well-suited for distributed multimedia systems (see [NS95] for an introduction) as well. After all, multimedia systems are real-time applications, so that most features of SSCMP should be suitable here anyway. Relaxing the guaranteed delivery of SSCMP to guarantees appropriate for multimedia transmission should be relatively straighforward. Of course, devising an appropriate resource management for guaranteed QoS (see [Kur93], [Tow93]) is a separate issue. However, even resource management tasks like monitoring of actual QoS may greatly be helped by synchronized clocks and timestamped messages as used in SSCMP.

2 Related Work

2.1 Connection Management Protocols

Connection-oriented protocols are important for several tasks in distributed computing, ranging from classical remote procedure calls and reliable data transfer up to transmission of multimedia data streams. Traditional connection management protocols as used in TCP (see [Pos81]) employ *initial handshaking* for setting up connection: To ensure that a message is not a duplicate, sender and receiver must *exchange* setup messages before actual data transmission can take place.

Initial handshaking is perfectly reasonable for infrequently setup but heavily used connections found in former generation computer networks, since the setup-overhead is effectively spread over all the messages sent via a connection. This is no longer true for the communication patterns found in today's (client-server) distributed systems, where frequent connections to numerous servers with only a few (often two) messages exchanged per connection are common, cf. [CW89]. In fact, the overhead of initial handshaking is one round-trip time, which is primarily determined by the signal propagation delay. It is

[4]Supporting industrial practice does of course not mean that we ignore the fact that the current state-of-the-art is largely based on unsuitable approaches, cf. [SR93]. However, in view of the incoherent state of research, we think that a *tabula rasa* philosophy —abandon everything in favour of something new— is inappropriate by now.

therefore not decreased by the dramatically increasing transmission speeds and becomes in fact more and more unbearable as high-speed network technology evolves.

Timer-based connection management protocols like the pioneering *Delta-t* ([Wat81]) avoid that connection setup overhead completely, providing a promising alternative. The basic idea underlying such protocols is to remember "recently" received messages in order to detect duplicated setups. This is made working by somehow enforcing a *maximum packet life/validity time*, and the few existing timer-based protocols differ in how this is actually accomplished. Any timer-based protocol relies on a common "idea" of time among all the nodes of the distributed system, and it has been realized early that such protocols may be both considerably enhanced and simplified by assuming that nodes are equipped with synchronized clocks; see [Che89], [LSW91], [BF93].

The pioneering *Delta-t* ([Wat81]), the first timer-based connection management protocol available, is a fully engineered transport protocol designed for system architectures without special hardware like synchronized clocks and stable storage. Connection records (abbreviated CR) are created "on demand" and become automatically (i.e., timer-based) released when it is guaranteed that no old packet is alive. This is made working by incorporating a *time-to-live* field into each message sent, which is appropriately decremented as the message travels through the network and intermediate nodes. To that end, some (reliable) link-transit-time protocol is required; the one presented in [Slo83] assumes that all node's clocks are running at approximately the same rate (although Delta-t does not need synchronized clocks). The receiver node retains a connection record just long enough to guarantee that any duplicate of a message generated during the lifetime of a connection has its time-to-live expired, exploiting the fact that any node that encounters a message with zero time-to-live must discard it.

Another fully engineered transport protocol relating to our SSCMP is *VMTP* described in [Che86], which employs a timer-based connection management scheme very similar to the one used in Delta-t. However, it exploits properties following from introducing T-stable addressing for duplication detection purposes also. In its original version, VMTP relied on a time-to-live field incorporated in each message; in its revised version (cf. [Che89]) it employs end-to-end timestamps to enforce maximum packet lifetimes. Hence, it requires synchronized clocks and also some sort of stable storage in order to generate T-stable identifiers valid even across node crashes.

A particularily carefully engineered and versatile "next generation" transport protocol is *XTP* described in [SDW92]. Targeted to high-speed networks, much emphasis has been laid upon easing implementation in hardware. However, as far as timer-based connection management is concerned, it does not provide novel ideas but relies on the mechanisms of Delta-t. Nevertheless, we should note that XTP actually uses a certain mixture of timer-based and handshake-based mechanisms, supporting a quick graceful close and hence speeding up connection release time. Note that the primary disadvantage of timer-based protocols with respect to handshake-based ones is the fact that a connection record is usually released later; nevertheless, even completely handshake-based protocols like TCP require some non-zero connection release time, cf. [Wat81].

The *CMSC* protocol described in [BF93] follows the approach underlying the revised VMTP ([Che89]) to remove the dependence of the protocol from the underlying network.

More specifically, an expiration time (instead of a time-to-live field) is added to each message, making the maximum packet lifetime enforcement purely end-to-end by means of synchronized clocks. Actual message transmission is governed by an ordinary sliding window protocol. Although CMSC is not a fully engineered protocol in the sense of Delta-t, VMTP, or XTP, it deals with a connection oriented interface in some detail.

Finally, our *Sequenced Synchronized Clock Message Protocol* as described in [SP95] employs messages carrying a timestamp and a sequence number in an integrated fashion. Connection records are only retained for some suitable period of time after transmission activities have ceased. However, the timestamp of the last message that arrived over a connection that is reclaimed is used to update a global upper bound that may be used for duplicate detection after releasing the connection record. Therefore, SSCMP requires synchronized clocks and some non-volatile memory.

Contrasting previous work ([Wat81], [Che86], [SDW92], [BF93]), which primarily addresses transport layer protocols supporting connection open, transfer, and close phases explicitly, we adhere to a both simplified and more abstract point of view, which hides away even the concept of connections from the *service specification* (cf. [Sha91]) of our SSCMP protocol: We view our protocol as being responsible for providing timely, reliable, sequenced, packet-oriented but otherwise unstructured data transmission on top of a necessarily imperfect communication system. Of course there are connections involved in the *protocol (entity) specification* of SSCMP, but they are internally managed and hence invisible from the outside. Note that this may be viewed as building a reliable datagram service. Thus, the usual dichotomy between connection oriented and datagram services is relaxed by SSCMP.

Our approach has several advantages: First of all, dealing with timer-based protocols invented to *avoid* connection setup overhead, we feel that adhering to a connection-based interface would —in some sense— give away some of those advantages. Moreover, relying on a low-level interface makes our basic protocol applicable for data link layer and transport layer protocols as well; constructing a proper transport layer interface dealing with important details like adressing is more or less straightforward. Last but not least, a protocol providing reliable, sequenced communication is a necessary basis for more advanced features, like atomic broadcast.

As far as correctness is concerned, SSCMP surpasses the core of any of the other protocols since it guarantees at-most-once delivery even in the case of clock synchronization failures. Note that a violation of the clock synchronization condition is a rather likely event, in particular in systems employing probabilistic algorithms (like NTP), but also in deterministic ones, where at least the possibility of faulty clocks exists. Unlike protocols like Delta-t, SSCMP also tolerates any (very) late message (arriving when the connection record has long been released), and its correctness does not depend on bounds on the transmission rate since there are no implicit timing constraints involved (as in sequence number wrapping/reuse or T-stability in other protocols). By the way, devising comprehensive correctness proofs might benefit from the fact that the SCMP protocol of [LSW91], the ancestor of our SSCMP, is amenable to formal verification, see [Lam93] for details.

Viewed from an "engineering perspective", one observes that SSCMP is purely end-to-end since it does not require support from the underlying network. Moreover, it allows for

multiple outstanding messages (beyond stop-and-wait), does not require static allocation of connection records, allows immediate resumption after reboot, and accepts messages generated during a long lasting receiver crash. Finally, its striking simplicity should be judged in view of the fact that packet losses in modern networks are mainly caused by receiver overruns, i.e., performance problems caused by complicated protocols, and not by transmission errors. It is understood, however, that SSCMP —unlike most of the abovementioned protocols— is of course not a fully engineered connection management protocol by now, in the sense that it does not deal with addressing and dozens of other "practical" issues.

To conclude, the following table provides a summary of the more detailled comparision of the protocols given in [SP95].

Protocol	Clock fault	Timer	CR.size	CR.release	Resumption	Complexity
Delta-t	not tolerated	3	small	2δ	not immediate	medium
VMTP	not tolerated	3	small	$> 2\delta$	immediate	medium
XTP	not tolerated	3	small	$(<)2\delta$	not immediate	medium
CMSC	not tolerated	3(5)	> small	2δ	immediate	high
SSCMP	tolerated	3	small	2δ	immediate	low

2.2 Multiplexing and Scheduling for Timely Delivery

Timely delivery of messages, i.e., delivery of messages by some specified deadline, is of course mandatory for any protocol designed for real-time applications. The problem to be solved here internally is how to multiplex (= schedule) packets originating from multiple connections in order to guarantee timely delivery for each connection. For fully connected (point-to-point) networks, this involves (multiple) classical, non-preemptive real-time scheduling problems.

In multi-access channels like Ethernet, the resource to be scheduled —the channel— is shared by all connections existing in the whole system, not only by the ones established at a single node. Needless to say, there is usually no centralized control of channel access. Therefore, we face a non-preemptive global scheduling problem that needs to be solved with locally available information. An overview of existing research may be found in [MZ95] and [KSY84].

Categorizing the work relevant for our purposes, it is important to note that we are going to follow the so-called "on-line school" in real-time systems design, which is a promising alternative to the well-established "off-line school". It is argued by prominent researchers in the field that it is the former one that will be able to satisfy the requirements of next generation real-time systems, cf. [LeL94], [HLR95], [SR93], for example. Actually, we do not see any reason to employ clearvoyancy assumptions underlying off-line approaches *when they are in fact not needed to provide a solution.* So why should we consider a protocol that works only for periodic/sporadic messages when there is one that can deal with fully aperiodic ones (in the sense that if messages are timely deliverable at all, the protocol will deliver them on time)?

Consequently, we will not consider the numerous papers dealing with guaranteed synchronous message communication, utilizing ideas like ordinary TDMA as in the TTP of

[KG94], combining rate monotonic scheduling with priority driven protocols as employed in the IEEE 802.5 token ring (see e.g. [SM89]), or synchronous bandwidth allocation schemes for the timed token protocol as used in FDDI, for example (see e.g. [ACZ93]). The same is true for approaches based on periodic servers for handling asynchronous (sporadic) messages (see [SML88]). It is most interesting to note that all these approaches have been shown to be inappropriate w.r.t. the abovementioned goals in [HLR95].

Also not appropriate for our purpose are most of the so-called *best-effort* schemes. Among them are virtual time protocols like the ones described in [ZR87], which try to approximate some (optimal) centralized scheduling algorithm like *minimum laxity first* (MLF) by CSMA/CD with selectively delayed initial channel access, and approaches that assume multiversion messages and employ some version selection scheme to control the amount of data transmitted, see e.g. [MZ91].

What we are looking for are algorithms suitable for implementing the abstraction of *real-time virtual circuits*[5] (RTVC, see [ARS91]), which guarantee timely delivery for all messages accepted for transmission. That is, when a message is submitted for transmission to a RTVC, there must be an admission test to check whether timely delivery is possible (without endangering previously guaranteed messages). If the admission test fails, the transmission request is rejected. Note that this approach is not at all inferior to off-line guarantees offered by static system designs: If message arrivals obey some (maximum) periodic/sporadic arrival law *a priori*, then no rejection will ever take place if proper algorithms are used. Suitable on-line guarantees are in fact superior to off-line ones, since they work even in the case when periodic/sporadic assumptions are violated.

The basic idea underlying suitable implementations of RTVCs is to adopt a deadline-driven real-time scheduling algorithm like *earliest deadline first* (EDF) or *minimum laxity first* (MLF) to select the next message that must be transmitted over the channel, in a distributed way. Synchronized clocks at all nodes of the distributed system are of course required for that purpose. A common notion of time, however, is not enough since the problem is primarily made difficult by the fact that knowledge about waiting messages is distributed among the nodes of the system.

There are basically two different ways to solve this problem, cf. [MZ95]: *dynamic reservation* and *conservative estimation*. The former method operates on system-wide reservation information made available locally by querying/informing other nodes for/of local reservation requests, see [MZB90], for example. Since local reservations must be observed consistently at all nodes, a synchonous reliable atomic broadcast (see Section 2.3) must be utilized for information dissemination if faults are to be considered. Although being time- and bandwidth-consuming, this method allows acceptance tests to be based on the actual system load.

Algorithms suitable for the alternative conservative estimation technique do not exchange reservation information. The algorithms employed here must admit worst case bounds on message delivery times (based on local information only) to support the acceptance test. This idea seems particular attractive to us, although it might reject messages

[5]Since SSCMP relaxes the usual dichotomy between virtual circuits (connections) and datagrams, the notion of real-time virtual circuits is somewhat unsatisfactory in our context —we have to deal with something like reliable real-time datagrams instead.

that could have been accepted.

For algorithms supporting conservative estimation, exchanging local information is usually replaced by exploiting the channel feedback of normal (message) transmissions. For example, in a bus network like Ethernet, a node starting transmission at time t learns from the occurence of a collision that some other node started its transmission as well. This little piece of global information is already sufficient to accomplish that in case of multiple simultaneous transmission attemps (originating at different nodes) only one node —that one that would have been selected by a fully centralized scheduler— (eventually) wins. Using channel feedback is of course a well-established technique for multiaccess channels, cf. [Gal85] for an excellent survey.

Several different methods how to use this technique in real-time communications have been proposed. Apart from (already "abandoned") virtual time algorithms like [ZR87], which try to avoid collisions by delaying a node's initial channel access according to its laxity or deadline, there is an efficient collision-free technique sometimes called *forcing headers* introduced in [RW77]. It exploits the inherent wired-or property of broadcast busses: If multiple transmitters transmit their messages starting with certain (priority-)field simultaneously, they will clash bit-for-bit. Therefore, only the transmitter with the highest priority value will read its priority information back from the channel; all others will observe feedback which is different from the value written out to the channel, forcing them to give up. Note that this technique is used in the MAC layer of the rapidly emerging CAN (*controller area network*) fieldbus.

There are also two interesting collision-based techniques, which should be suitable for our purposes. *Window protocols* are based on adopting the idea underlying the splitting algorithm of Gallager / Tsybakov and Mikhailov (cf. [Gal85]) to work on latest-time-to-send (LTS) instead of message arrival times. More specifically, collisions are resolved by maintaining a (consistent) time window at each node, which governs transmission rights of a node. If a node's LTS is within the current window, it may transmit, otherwise it has to remain silent. If a collision occurs, the window is split and only transmitters having their LTS in the lower half may proceed. Splitting is repeated until a successful transmission takes place. Therefore, window protocols like the one described in [Zua91], [ZSR90] effectively implement MLF scheduling.

A promising alternative to window protocols is based on adopting *Deterministic Ethernet* ([BFR87], [LeL87]) to implement EDF scheduling. This DOD-CSMA-CD (*deadline-oriented deterministic CSMA-CD* protocol introduced in [LR93] is particularily interesting, because (1) EDF was shown recently to be optimal not only for preemptive but also for *non-preemtive* tasks, see [GMR95], and (2) EDF does not require *a priori* knowledge of message transmission times as does MLF. Moreover, there is a complete worst case analysis of the protocols' behaviour that provides a simple acceptance test based on conservative estimation.

2.3 Atomic Broadcast

Atomic broadcast[6] is one of the most central issues in fault-tolerant computing. For example, if servers (e.g. file servers) are replicated to increase reliability and availability of operation, it is usually crucial that all replicas obtain all the service requests (e.g. transaction messages) *consistently*; otherwise, replica inconsistency might occur. Moreover, reliable delivery must be ensured even when some system components fail. If fault-tolerant real-time applications are to be considered, it must be guaranteed that any broadcast terminates successfully within some known time. Note that timely transmission multiplexing and channel scheduling is a neccessary prerequisite here.

There are several different types of reliable broadcast[7], imposing certain restrictions to the order messages are received by different servers. *Reliable broadcast* is the weakest form, allowing arbitrary order of reception. *FIFO broadcast* guarantees that messages broadcast by the same sender are delivered in the order they were broadcast. If broadcasts of two senders are causally related, the strongest *causal broadcast* ensures that the reception order respects the causality relation. Protocols implementing causal order were published by [BJ87], [PBS89], [SES89], [BSS91], [SB91], [FT92] and [APR93]. Note that causal broadcast is also easily provided on top of FIFO broadcast, cf. [HT93]. Finally, *atomic broadcast* guarantees that receptions at all receivers are in exactly the same order; *FIFO atomic broadcast* and *causal atomic broadcast* combine the appropriate constraints.

Two classes of protocols for atomic broadcast have been proposed to date: *asynchronous protocols*, which use message ackowledgements, and *synchronous protocols*, which rely on synchronized clocks to enforce total order by means of chronological order (of course consistent with causality). In the following subsections, we give a brief survey of existing broadcast specifications and protocols.

2.3.1 Asynchronous Protocols

Though being implemented quite differently, all varieties of (asynchronous) atomic broadcast protocols guarantee at least the following properties:

- *Agreement*: All correct nodes agree on the set of messages they deliver.
- *Validity*: All messages broadcast by correct nodes are delivered.
- *Order*: All correct nodes deliver messages in the same unique order, even though this order is not determined in advanve.

Among the protocols that claimed to provide this service are the protocols introduced in [CM84] for Ethernet or satellite networks, differing mainly in the degree of fault-tolerance that they provide. Their algorithms work centralized: All sources transmit to a central site, called the token site, which assigns sequence numbers to the messages

[6]This communication service is variously termed reliable [CM84] or atomic [CAS+85]. We will refer to it as atomic broadcast.

[7]Usually, one distinguishes *broadcast* addressing all nodes of a distributed system from *multicast* addressing all nodes within a certain process group. The major difference comes from the fact that managing process groups requires a membership protocol. We will not deal with managing membership within the SSCMP-project, so multicast and broadcast are used synonymously.

and forwards them to the destination sites. An elegant token-passing protocol is used to detect failures at the token site, to select a new token site, and to retransmit messages affected by the failure. This early protocol inspired the research of Garcia-Molina and Spauster, see [GS89], [GS91]. Instead of ordering all messages at a central side, their algorithm orders them by a collection of nodes structured into a message propagation graph. The graph indicates the paths messages should follow to get to all intended destinations. Instead of sending the messages to the destinations and then ordering them, the messages get propagated via a series of sites that order them along the way by merging messages destined for different groups. The protocol described in [KTH+89] resembles the protocol of [CM84] that cannot recover from processor crashes, yet is optimized for the common case of no communication failures. The design of a simple reliable totally-ordered broadcast service (cf. [Oes91]) is based on [KTH+89] and implemented on a standard Unix system using the standard TCP/IP protocol family.

The ISIS distributed programming environment (cf. [BJ87], [BJ88]) supports a wide range of multicast protocols from causal orderings (*CBCAST*) to total orderings (*ABCAST*) relying on two-phase commit mechanisms. Each site maintains a priority queue per process. The sender multicasts the message to the various destinations which each assign it their own priority number. The message is marked "undeliverable" and put on the queue. Each receiver returns the priority number to the sender, which in turn picks out the highest one and sends it back to the receivers. The receivers replace their original numbers with the new one and tag the message as "deliverable". After reordering a message may be delivered if it is the first one in the queue. Dasser showed in [Das92] how this protocol may even be enhanced by reducing the latency time between marking the message as "deliverable" and its delivery. Finally, the "second generation" of ISIS protocols (cf. [BSS91]) supports highly concurrent applications and scales to systems with large numbers of potentially overlapping process groups.

Quite close in the concept to Birman's and Joseph's approach are the *Trans* and *Total* protocol described in [MMA90] and the *Psync* protocol introduced in [PBS89]. The *Trans* protocol piggybacks acknowledgments for old broadcast messages on new ones and guarantees that all processes eventually construct the same partial order of broadcast messages. The *Total* protocol is even able to establish a consistent total order. However the efficiency of *Trans* and *Total* depends on the use of a broadcast communication medium. *Psync* is a low-level protocol designed to support a variety of high-level protocols and distributed applications. It maintains only the partial order among messages represented in the form of a direct acyclic graph, called the context graph. A collection of "higher" routines which may be applied on this graph enforces various ordering disciplines. Motivated by the *Trans* algorithm and the *Psync* algorithm [ADK+92] provide services for membership and so called basic-, causal-, agreed- and safe-multicast resembling the ISIS approach, however differing in design and implementation.

Luan and Gligor [LG90] devised a promising protocol based on a variation of three-phase commit that uses voting to avoid blocking. Finally, Nakamura and Takizawa presented a cluster concept —an extension of the conventional connection concept to multiple service access points (SAPs)— and built a selectively partially ordering protocol *SPO* (cf. [NT91]) and a totally-ordering protocol *TO* (cf. [TN89]) on top of it that works under

distributed control.

2.3.2 Synchronous Protocols

The atomic broadcast protocols mentioned so far use message acknowledgements to ensure the listed properties. However, they cannot guarantee *timeliness*: If a message is delivered at all, it has to be delivered within a bounded time after it was broadcast. Thus, synchronous protocols relying on a round-based model, can be used to update *synchronously* a distributed storage that displays the same contents at every correct process. They ensure the existence of a time constant Δ such that the following properties are satisfied:

- *Agreement and Validity*: If any node processor delivers an update by time U on its clock, then that update was initiated by some node and is delivered by all correct nodes by time U on their clocks.
- *Order*: All correct nodes deliver messages in the same order.
- *Termination*: Every broadcast initiated by a correct node at time T on its clock is delivered by all correct nodes by time $U = T + \Delta$ on their clocks.

Synchronous broadcast protocols may be applied for critical real-time applications which must enforce bounds on response times even when failures occur. These failures may be classified in increasing severity as follows (cf. [BB93]):

- *Crash failure*: A component stops participating in the protocol prematurely.
- *Omission failure*: A component fails to send some of its messages.
- *General omission failure*: A component fails to send or receive some messages.
- *Timing failure*: A component either omits to respond or responds too early or too late.
- *Authenticated byzantine failure*: A component shows arbitrary behavior but there is a message authentication scheme that prevents the component from forging the messages of correct components.
- *Byzantine failure*: A component shows arbitrary behavior.

The problem of atomic broadcast in the presence of faults was first studied by Christian et al. [CAS+85] (see also [CDS+90]). They developed synchronous protocols proposed for point-to-point networks managing failures up to authenticated byzantine failures. The protocols are based on simple message forwarding: a processor forwards any new message it receives from a link on all other links as soon as it receives the message. The main drawback of this prompt forwarding technique is that message forwarding always takes place even when no failures occure. Improvements of their protocols proposed for redundant broadcast channels that do not forward messages when there is no need to do (following the lazy forwarding rule) may be found in [Cri90].

[BD85], [BSD88] describe a protocol based on message rounds and exactly synchronized clocks in which all receiving processors know the time a sending processor broadcasts. They studied the execution time of their protocol for any communication graph.

including familiar structures such as fully connected point-to-point graphs, rings, busses and broadcast networks, and obtained lower bound results identifying a time gap between systems where processors may only fail to send messages, and systems where processors may fail both to send and to receive messages.

Within the project Mars [KG94] developed and implemented a practical multicast protocol for time-triggered architectures that uses a TDMA broadcast medium with simple algorithms and low overhead. Based on the assumption that the communication channel have only omission failures and that the nodes support the fail silent abstraction their protocol TTP integrates services like message transport with predictable latency, membership service and support for rapid mode change.

Last but not least the work of [VRB89], [VM90] should be mentioned, which occupies a special position between the two main approaches (i.e., asynchronous and synchronous protocols) using properties of the underlying network to enforce known and bounded execution times without the existence of a global clock.

2.4 Synchronized Clocks

Our previous expositions should have made clear that synchronized clocks are required for reliable (FIFO) data transmission (connection management), timely delivery, and atomic broadcast. Actually, it is well known that a common notion of time among the nodes of a distributed system greatly simplifies the design of most distributed services, see [Lis93] for an overview. Combining this fact with the trend towards integrating (large) distributed systems into daily life, which is governed by *universal time coordinated* (UTC), we think that it is not unreasonable to predict that future generation computer systems will be equipped with accurately synchronized clocks.

In fact, much effort has been devoted to the development of (inexpensive) techniques for clock synchronization (see [SWL90], [RSB90] for an overview and [YM93] for a bibliography), and high-accurate, inexpensive sources of (UTC) are worldwide available now via the NAVSTAR *global positioning system* GPS (see [Wel87]). The development of the *network time protocol* NTP (see [Mil91]) has pushed synchronized clocks even into Internet-reality. Last but not least, there is our well-advanced research project SynUTC, which will provide hardware and software implementing high-accuracy synchronized clocks by incorporating GPS receivers, see [Sch95] for related issues. The results of SynUTC are of course ideally suited to be used as a basis for the SSCMP-project.

2.5 Selected Bibliography

References

[ACZ93] G. Agrawal, B. Chen, W. Zhao. *Local Synchronous Capacity Allocation Schemes for Guaranteeing Message Deadlines with the Timed Token Protocol*, Proc. INFOCOM '93, 1993, p. 186–193.

[ADK+92] Y. Amir, D. Dolev, S. Kramer, D. Malki. *Transis: A Communication Sub-System for High Availability*, 20th FTCS, 1992, p. 76–84.

[APR93] R. Aiello, E. Pagani, G. Rossi. *Causal Ordering in Reliable Group Communications*, SIGCOMM'93, 1993, p. 106–115.

[ARS91] K. Arvind, K. Ramamritham, J.A. Stankovic. *A Local Area Network Architecture for Communication in Distributed Real-Time Systems*, Real-Time Systems, 3(2), 1991, p. 115–147.

[BB93] P. Berman, A. Bharali. *Quick Atomic Broadcast*, Distributed Algorithms, 1993, p. 189–203.

[BD85] Ö. Babaoglu, R. Drummond. *Streets of Byzantium: Network Architectures for Fast Reliable Broadcasts*, IEEE Transactions on Software Engineering, SE-11(6), 1985, p. 546–554.

[Bel76] D. Belsnes. *Single-Message Communication*, IEEE Transactions on Communications, COM-24(2), February 1976, p. 190–194.

[BF93] E. Biersack, D. Feldmeier. *A timer-based connection management protocol with synchronized clocks and its verification*, Computer Networks and ISDN Systems, 25, 1993, p. 1303–1319.

[BFR87] J. Boudenant, B. Feydel, P. Rolin. *An IEEE 802.3 Compatible Deterministic Protocol*, Proc. IEEE Infocom '87, 1987, p. 573–579.

[BJ87] K. Birman, T. Joseph. *Reliable Communication in the Presence of Failures*, ACM Transactions on Computer-Systems, 5(1), 1987, p. 47–76.

[BJ88] K. Birman, T. Joseph. *Reliable Broadcast Protcols*, Lectures Notes of Artic 88, 1988.

[BSD88] Ö. Babaoglu, P. Stephenson, R. Drummond. *Reliable Broadcasts and Communication Models: Tradeoffs and Lower Bounds*, Distributed Computing, 2, 1988, p. 177–189.

[BSS91] K. Birman, A. Schiper, P. Stephenson. *Lightweight Causal and Atomic Group Multicast*, ACM Transactions on Computer Systems, 9(3), 1991, p. 272–314.

[CAS+85] F. Cristian, H. Aghili, R. Strong, D. Dolev. *Atomic Broadcast: From Simple Diffusion to Byzantine Agreement*, 15th FTCS, Ann Arbor, Michigan, 1985.

[CDS+90] F. Cristian, D. Dolev, R. Strong, H. Aghili. *Atomic Broadcast in a Real-Time Environment* in: S. Simons, A. Spector (ed.). *Fault-Tolerant Distributed Computing*, Springer Verlag, 1990, p. 51–71.

[Che86] D. Cheriton. *VMTP: A Transport Protocol for the Next Generation of Communication Systems*, Proc. SIGCOMM '86, 1986, p. 406–415.

[Che89] D. Cheriton. SIRPENT^{TM}: *A High-Performance Internetworking Approach*. Proc. SIGCOMM '89, Austin, Texas, September 1989, p. 158–169.

[CM84] J. Chang, N. Maxemchuk. *Reliable Broadcast Protocols*, ACM Transactions on Computer Systems 2(3), 1984, p. 251–273.

[Cri90] F. Cristian. *Synchronous Atomic Broadcast for Redundant Broadcast Channels*, Journal of Real-Time Systems, 2, 1990, p. 195–212.

[CW89] D. Cheriton, C. Williamson. *VMTP as the Transport Layer for High-Performance Distributed Systems*, IEEE Communications Magazine, June 1989, p. 37–44.

[Das92] M. Dasser. *TOMP - A Total Ordering Multicast Protocol*, Operating Systems Review, 26(1), 1992, p. 32–40.

[FT92] G. Florin, C. Toinard. *A New Way to Design Causally and Totally Ordered Multicast Protocols*, Operating Systems Review, 26(4), 1992, p. 77–83.

[Gal85] G. Gallager. *A Perspective on Multiaccess Channels*, IEEE Transactions on Information Theory, IT-31(2), 1985, p. 124–142.

[GMR95] L. George, P. Muhlethaler, N. Rivierre. *Optimality and Non-Preemptive Scheduling Revisited*, INRIA Research Report no. 2516, April 1995.

[GS89] H. Garcia-Molina, A. Spauster. *Message Ordering in a Multicast Environment*, Proc. 9th International Conference on Distributed Computing Systems. 1989, p. 354–361.

[GS91] H. Garcia-Molina, A. Spauster. *Ordered and Reliable Multicast Communication*, ACM Transactions on Computer Systems, 9(3), 1991, p. 242–271.

[GT91] A. Gopal, S. Toueg. *Inconsistency and Contamination*, Proc. 10th ACM Annual Symposium on Principles of Distributed Computing, Montreal, 1991, p. 257–272.

[HLR95] J.-F. Hermant, G. Le Lann, N. Rivierre. *A General Approach to Real-Time Message Scheduling over Distributed Broadcast Channels*, Proc. INRIA/IEEE Conference on Emerging Technologies and Factory Automation, October 1995, Paris.

[Hor94] M. Horauer. *Entwicklung einer Network Timestamp Unit für einen Versatile Timing Analyzer zum Monitoring von verteilten Echtzeitsystemen*, Diplomarbeit Dept. of Computer Technology, Technical University of Vienna, 1994. (in german)

[HT93] V. Hadzilacos, S. Toueg. *Fault-Tolerant Broadcasts and Related Problems*, in: S. Mullender (ed.). *Distributed Systems*, 2nd ed., Addison Wesley, 1993. p. 97–145.

[JL87] P. Jain, S. Lam. *Modelling and Verification of Real-Time Protocols for Broadcast Networks*, IEEE Transaction on Software Engineering, SE-13(8), 1987, p. 924–937.

[KG94] H. Kopetz, G. Grünsteidl. *TTP — A Protocol for Fault-Tolerant Real-Time Systems*, IEEE Computer, January 1994, p. 14–23.

[KSY84] J. Kurose, M. Schwartz, Y. Yemini. *Multiple-Access Protocols and Time Constrained Communication*, ACM Computing Surveys, 16(1), 1984, p. 43–70.

[KTH+89] F. Kaashoek, A. Tanenbaum, S. Hummel, H. Bal. *An Efficient Reliable Broadcast Protocol*, Operating Systems Review, 23(4), 1989, p. 5–19.

[Kur93] J. Kurose. *Open Issues and Challanges in Providing Quality of Service Guarantees in High-Speed Networks*, ACM Computer Communication Review, 23(1), January 1993, p. 6–15.

[Lam93] B. Lampson. *Reliable Messages and Connection Establishment*, in: S. Mullender (ed.). *Distributed Systems*, 2nd ed., Addison-Wesley, 1993, p. 251–281.

[LG90] S. Luan, V. Gligor. *A Fault-Tolerant Protocol for Atomic Broadcast*, IEEE Transactions on Parallel and Distributed Systems, 1(3), 1990, p. 271–285.

[LeL87] G. Le Lann. *The 802.3 D Protocol: A Variation on the IEEE 802.3 Standard for Real-Time LANs*, INRIA Technical Report, 1987.

[LeL94] G. Le Lann. *Scheduling in Critical Real-Time Systems: a Manifesto*, Proc. 3rd Int. Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, Germany, September 1994, Springer LNCS 863.

[Lis93] B. Liskov. *Practical uses of synchronized clocks in distributed systems*, Distributed Computing, 6, 1993, p. 211–219.

[Loy95] D. Loy. *GPS-Linked High Accuracy NTP Time Processor for Distributed Fault-Tolerant Real-Time Systems*, Dissertation Faculty of Electrotechnics, Technical University of Vienna, 1995. (forthcoming)

[LR93] G. Le Lann, N. Rivierre. *Real-Time Communications over Broadcast Networks: the CSMA-DCR and the DOD-CSMA-CD Protocols*, INRIA Rapport de Recherche 1863, 1993.

[LSW91] B. Liskov, L. Shrira, J. Wroclawski. *Efficient At-Most-Once Message Based on Synchronized Clocks*, ACM Transactions on Computer Systems, 9(2), May 1991, p. 125–142.

[Mil91] D. Mills. *Internet Time Synchronization: The Network Time Protocol*, IEEE Transactions on Communications, 39(10), October 1991, p. 1482–1493.

[MMA90] P. Melliar-Smith, L. Moser, V. Agrawala. *Broadcast Protocols for Distributed Systems*, IEEE Transactions on Parallel and Distributed Systems, 1(1), 1990, p. 17–25.

[MMSa] MUMM e.V.: *Basic M-Modules Specification*, Nürnberg, Germany.

[MMSb] MUMM e.V.: *M-Module Directory*, Nürnberg, Germany, August 1993.

[Mul93] S. Mullender. *Interprocess Communication*, in: S. Mullender (ed.), *Distributed Systems*, 2nd ed., Addison-Wesley, 1993, p. 217–250.

[MZ91] N. Malcolm, W. Zhao. *Version Selection Schemes for Hard Real-Time Communications*, Proc. IEEE Real-Time Systems Symposium, 1991, p. 12–21.

[MZ95] N. Malcolm, W. Zhao. *Hard Real-Time Communication in Multiple Access Networks*, Real-Time Systems, 8, 1995, p. 35–77.

[MZB90] N. Malcolm, W. Zhao, C. Barter. *Guarantee Protocols for Communication in Distributed Real-Time Systems*, Proc. IEEE Infocom '90, 1990, p. 1078–1086.

[NS95] K. Nahrstedt, R. Steinmetz. *Resource Management in Networked Multimedia Systems*, IEEE Computer, May 1995, p. 52–63.

[NT91] A. Nakamura, M. Takizawa. *Reliable Broadcast Protocol for Selectively Partially Ordering PDUs (SPO Protocol)*, COMPSAC'91, 1991, p. 239–246.

[Oes91] D. Oestreicher. *A Simple Reliable Globally-Ordered Broadcast Service*, Operating Systems Review, 25(4), 1991, p. 66–76.

[PBS89] L. Peterson, N. Buchholz, R. Schlichting. *Preserving and Using Context Information in Interprocess Communication*, ACM Transactions on Computer Systems, 7(3), 1989, p. 217–246.

[Pos81] J. Postel. *DoD Standard Transmission Control Protocol*, DARPA-Internet RFC 793, 1981.

[Pus95] A. Pusterhofer. *NTSU Network Timestamp Unit: Ein Modul für Eventtimestamping und Uhrensynchronisation via LAN*, Diplomarbeit Dept. of Automation 183/1, Technical University of Vienna, 1995. (in german)

[RSB90] P. Ramanathan, K. Shin, R. Butler. *Fault-Tolerant Clock Synchronization in Distributed Systems*, IEEE Computer, 23(10), October 1990, p. 33–42.

[RW77] E. Rothauser, D. Wild. *MLMA-A Collision-Free Multi-Access Method*, Proc. IFIP Congress, 1977, p. 431–436.

[SB91] P. Stephenson, K. Birman. *Fast Causal Multicast*, Operating Systems Review, 25(2), 1991, p. 75–80.

[Sch95] U. Schmid. *Synchronized Universal Time Coordinated for Distributed Real-Time Systems*, Control Engineering Practice, 3(6), 1995, p. 877–884.

[SDW92] W. Strayer, B. Dempsey, A. Weaver. *XTP — The Xpress Transfer Protocol*, Addison Wesley, 1992.

[SES89] A. Schiper, J. Eggli, A. Sandoz. *A New Algorithm to Implement Causal Ordering*, Proceedings of the 3th International Workshop on Distributed Algorithms, 1988.

[Sha91] A. Shankar. *Modular Design Principles for Protocols with an Application to the Transport Layer*, Proceedings of the IEEE, 79(12), December 1991, p. 1687–1707.

[SLL93] J. Sogaard-Andersen, N. Lynch, B. Lampson. *Correctness of Communication Protocols: A Case Study*, MIT/LCS/TR-589, November 1993.

[Slo83] L. Sloan. *Mechanisms that Enforce Bounds on Packet Lifetimes*, ACM Transactions on Computer Systems, 1(4), November 1983, p. 311–330.

[SM89] J. Strosnider, T. Marchok. *Responsive, Deterministic IEEE 802.5 Token Ring Scheduling*, Real-Time Systems 1(2), 1989, p. 133–158.

[SML88] J. Strosnider, T. Marchok, J. Lehoczky. *Advanced Real-Time Scheduling Using the IEEE 802.5 Token Ring*, Proc. IEEE Real-Time Systems Symposium, 1988, p. 42–52.

[SP95] U. Schmid, A. Pusterhofer. *SSCMP: The Sequenced Synchronized Clock Message Protocol*, to appear in Computer Networks and ISDN Systems, 1995. (23 pages).

[SR93] J. Stankovic, K. Ramamritham. *Advances in Real-Time Systems*, IEEE Computer Society Press, 1993, p. 1–25.

[SS95] K. Schossmaier, U. Schmid. *UTCSU Functional Specification*, Technical Report Dept. of Automation, Technical University of Vienna, no. 1-56, 1995.

[SWL90] B. Simons, L. Lundelius-Welch, N. Lynch. *An Overview of Clock Synchronization*, B. Simons, A. Spector, editors: Fault-Tolerant Distributed Computing, Lecture Notes on Computer Science 448, 1990, p. 84–96.

[Tan81] A. Tanenbaum. *Computer Networks*, Prentice Hall, 1981.

[TN89] M. Takizawa, A. Nakamura. *Totally Ordering Broadcast (TO) Protocol on the Ethernet*, Proceedings of the IEEE Pacific RIM Conference on Communications, Computers and Signal Processing, 1989, p. 357–364.

[TN89] M. Takizawa, A. Nakamura. *Totally Ordering Broadcast (TO) Protocol on the Ethernet*, Proceedings of the IEEE Pacific RIM Conference on Communications, Computers and Signal Processing, 1989, p. 357–364.

[Tow93] D. Towsley. *Providing Quality of Service in Packet Switched Networks*, Proc. Joint Conference Performance'93 and Sigmetrics'93, Lecture Notes on Computer Science no. 729, Springer, 1993, p. 560–586

[Tur93] K. Turner. *Using Formal Description Techniques: An Introduction to ESTELLE, LOTOS and SDL*, Wiley Series in Communication and Distributed Systems, 1993.

[VM90] P. Verissimo, J. Marques. *Reliable Broadcast for Fault-Tolerance on Local Computer Networks*, 20th FTCS, 1990, p. 54–63.

[VRB89] P. Verissimo, L. Rodrigues, M. Baptista. *AMp: A Highly Parallel Atomic Multicast Protocol*, SIGCOMM'89, 1989, p. 83–93.

[Wat81] R. Watson. *Timer-Based Mechanisms in Reliable Transport Protocol Connection Management*, Computer Networks, 5, 1981, p. 47–56.

[Wel87] D. Wells. *Guide to GPS Positioning*, Canadian GPS Associates, 1987.

[YM93] Z. Yang, T. Marsland. *Annotated Bibliography on Global States and Times in Distributed Systems*, ACM SIGOPS Operating Systems Review, 27(3), July 1993, p. 55–72.

[ZH95] P. Zhou, J. Hooman. *Formal Specification and Compositional Verification of an Atomic Broadcast Protocol*, Real-Time Systems, 9, 1995, p. 119–145.

[Zna91] P. Znati. *Deadline-Driven Window Protocol for Transmission of Real-Time Traffic*, Proc. 10th IEEE International Conference on Computers and Communications, 1991, p. 667-673.

[ZSR90] W. Zhao, J. Stankovic, K. Ramamritham. *A Window-Protocol for Transmission of Time-Constrained Messages*, IEEE Transactions on Computers, 39(9), 1990, p. 1186–1203.

[ZR87] W. Zhao, K. Ramamritham. *Virtual Time CSMA Protocols for Hard Real-Time Communications*, IEEE Transactions on Software Engineering, SE-13(8), 1987, p. 938–952.

3 Project Definition

In this section, we will describe the envisioned goals of the SSCMP project and the intended ways of approaching them. It is our purpose to provide a fully engineered implementation of our SSCMP protocol for several different classes of networks, including a complete theoretical and also experimental evaluation of the protocol's properties. The implementation of SSCMP will consist of software in conjunction with elaborate hardware support built on top of existing technology. Therefore, the project should be classified as *oriented basic research* with strong emphasis on immediately applicable results.

3.1 Project Goals and Basic Approaches

Basically, there are four different (but obviously intimately related) tasks to be performed in SSCMP, which are described in the following subsections.

3.1.1 Concepts and Implementation (Software)

Numerous issues are to be considered in order to develop a fully-engineered protocol providing timely delivery and atomic broadcast on top of the novel connection management protocol of [SP95]. The most important issues are as follows.

- *Improving flow/rate control*

  There are several ideas of how to improve the SSCMP-variant presented in [SP95]. For example, SSCMP is modular in the sense that different protocols for flow/rate control may be plugged in; the existing variant uses a sliding window protocol (see [Tan81]) for that purpose. The underlying acknowledgment-based flow-control scheme, however, is inappropriate for very high speed networks. Rate control or credit-based algorithms should be employed here, cf. [SDW92].

- *Connection duration agreement*

  One topic should be to find improved strategies for connection record expiration and (re-)initialization; the ideas used in the SSCMP of [SP95] are not as robust as we would like under exceptional circumstances. There should also be a way for the application to control how long SSCMP holds its externally invisible connection records; this might include measures for a quick graceful close as used in XTP, cf. [SDW92]. Note that keeping connection records unnecessarily long wastes (non-volatile) memory; releasing it too early may cause frequent connection record (de)allocations to take place. However, care (i.e., multiple *R.upper* instances, see [SP95]) must be taken to avoid unrelated connection releases to influence each other.

- *Bidirectional communication*

  To improve the protocol's performance (in particlar for low-capacity networks), piggybacked acknowledgement techniques must be devised to reduce network load.

- *Implementing real-time virtual circuits (RTVCs)*

  We must provide (1) a suitable algorithm for multiplexing multiple connections at a single sender and (2) a proper channel scheduling algorithm for the broadcast channels considered to implement RTVCs as introduced in Section 2.2. This will eventually guarantee timely delivery of messages transmitted by SSCMP. Of course, SSCMP is exceptionally suitable to be used with all varieties of deadline-driven scheduling algorithms (like *earliest deadline first*), since the protocol depends on *timestamping* of messages.

  In principle, we think that forcing header protocols, window protocols, and DOD-CSMA-CD are all suitable for providing timely delivery in SSCMP. However, it calls for considerable research efforts to make those algorithms really applicable, e.g., as fault-tolerant as they should be. For example, the latter two (collision-based) protocols require that all nodes —even idle ones— maintain a consistent view of the transmission window; therefore, collision resolution may fail if a station wrongly senses a collision as idleness or successful transmission.

  Observe that we are aiming at a uniform approach, that is, something which may be adopted for very different networks, ranging from fieldbusses up to very high-speed networks. This implies research on improved algorithms, in particular modifying the forcing header scheme employed in CAN in order to make it suitable for guaranteed delivery. Finally, much theoretical research has to be done to establish sound acceptance tests à la DOD-CSMA-CD for the other algorithms.

- *Atomic broadcast*

  It is our purpose to build a timely FIFO or maybe causal atomic broadcast protocol on top of the connection management core of SSCMP. Although it is primarily the novel reliable connection management (that is, data transmission) protocol and not the atomic broadcast algorithm employed that we think will contribute to improving the state of the art, there may be room for novel results in the latter area as well. Note that the timer-based structure of our connection management protocol supports $1 : n$ transmissions automatically, probably opening up new ways of handling the acknowledgment problem.

  In order to achieve sufficient fault-tolerance, redundant broadcast channels are usually mandatory, cf. [Cri90], [BD85] for example. Again, the basic protocols' timer-based duplication detection feature is capable of handling redundant channels automatically. However, we have to be careful to avoid problems with channels/senders exhibiting byzantine faulty behaviour, cf. [CAS+85]. To that end, we should consider providing message authentication (i.e., encryption) at low (hardware) level, cf. Section 3.1.2.

- *Protocol interface*

  Since SSCMP will provide service not normally found in standard protocols, we need a non-standard interface to its functionality. Remember that SSCMP relaxes the usual distinction of connection oriented (COS) and connectionless services (CLS) by providing something like a timely reliable connectionless (datagram) service.

Interfaces built upon real-time variants of COS and CLS, as proposed in [ARS91], are therefore not really appropriate for our purposes. Note that we will also provide services required for network maintenance and testing, something that is required badly in practice.

Nevertheless, in order to being able to use SSCMP as the basis of a TCP/IP protocol stack, we should provide a standard 802.x (data link layer) interface for our protocol as well.

3.1.2 Concepts and Implementation (Hardware)

We think that the protocols' full advantages show up only with hardware support, in particular in case of (very) high speed networks. The following issues call for being implemented in an ASIC (possibly in conjunction with standard devices, e.g. dedicated processors and memories):

- *Connection identifier $\Longrightarrow$ connection record mapping*

  SSCMP does not provide the notion of a connection-oriented service in its service specification. Moreover, it does not need statically allocated storage for connection records but allows dynamic allocation/deallocation (based on the progress of time) instead. This opens up the possibility to use "real" datagram messages, carrying full (logical) sender/receiver identifiers instead of dynamically allocated connection identifiers. This "stateless" approach should considerably simplify the design of the protocol and also improve robustness and performace in case of faults.

  To exploit this property, there is a need of mapping sender/receiver identifiers to the appropriate connection records in real-time. This may be done either by using some special virtual memory management technique or by hashing. In any case, locating the connection record is necessary every time a message is submitted to the protocol for transmission, and every time when a message is received. That is, mapping must keep pace with data transmission speeds, making hardware support mandatory.

- *Monotonicity enforcement for sequenced timestamps*

  SSCMP is based on an integrated representation of timestamps and sequence numbers, called sequenced timestamps. In [SP95], monotonicity of sequenced timestamps —even across node crashes— has been shown to be a necessary precondition for correctness of the protocol. Of course, sequenced timestamps must be generated every time a message is to be transmitted. Therefore, generating monotonic sequenced timestamps should be performed in hardware.

- *Stable storage for connection records*

  Uncorrupted connection records are of vital importance for correctness of the protocol. Since the performance of SSCMP in case of crashes is considerably improved when connection records for existing connections survived the crashes (although this is not mandatory, see the next item), it is advantageous to organize memory for connection records as a stable storage built upon non-volatile RAM. To that end,

checksums (like CRC) for data written into memory are required. Given the quite frequent memory accesses, this must be done in hardware.

- *Maintaining reinitialization information*

  There is a simple way of reinitializing SSCMP to a safe state after a catastrophic crash where connection records get corrupted. This requires only a single data item (called *R.latest* in [SP95]), which must be kept in stable storage. Since it experiences frequent periodical updates, this should be done in hardware.

Of course, any specialized hardware support for SSCMP must be integrated with the network controller that implements the lowest layer(s) of the network protocol. Moreover, in order to provide accurately synchronized clocks, the UTCSU-ASIC developed in our SynUTC project should be utilized. Fortunately, there is a relatively efficient way of accomplishing this: We will develop a revised version of the *Network Clock Synchronization Coprocessors* (NCSC) developed in the SynUTC-project that adds the SSCMP support mentioned above.

Building an NCSC does of course not mean just assembling some standard network controller chipsets with our SSCMP- and UTCSU-ASIC. The following issues need careful consideration:

- *Encryption*

  Contrasting ISO practice, there are arguments in favour of providing encryption at the data link layer in order to being able to reject deliberately injected packets upon decryption, cf. [Mul93].

- *Multiplexing and channel scheduling for timely delivery*

  Dealing with timely transmission usually requires much support from the MAC layer, remember Section 2.2. One should carefully consider whether it is possible to implement such mechanisms on top of existing (programmable) controllers like the 68EN360; otherwise, development of VLSI network controllers from the scratch is inevitable.

- *Functional addressing and demultiplexing*

  To increase flexibility of a distributed system, one should avoid exporting node internals where possible (information hiding). Therefore, functional addressing is often used to accomplish that a client must not know the actual process ID of a server process it wants to connect to. Of course, some sort of mapping of the (usually large) domain of functional IDs to actual process IDs upon reception is required here. More yet, the network controller should be capable of demultiplexing based on functional IDs: A packet should be received into a buffer that can be mapped quickly (without copying) into the virtual address space of the appropriate (receiving) process, cf. [Mul93]. Note that those features might be dealt (or at least integrated) with the connection record mapping support mentioned earlier.

- *Maintenance services*

One should consider to provide (or, at least, exploit) maintenance features like *time domain reflectometry* for locating breaks/short circuits in network cabling, and support diagnosis by a promiscuous reception mode, for example. Such features are often provided by state-of-the-art chipsets, like Intel's 82596 Ethernet controller. Moreover, special attention should be laid upon plug&play features (auto-configuration, live-insertion, etc.).

In order to assess SSCMP's suitability for networks with totally different capabilities, we are planning to develop at least two different NCSC's:

- (dual/triple redundant) NCSC-CAN for the CAN-bus
- (dual/triple redundant) NCSC-Ethernet (possibly a fast, e.g., 100 Mbit/sec variant)

Given our basic approach to timely message delivery outlined in Section 3.1.1, there is no real alternative to the CAN bus; note that it has been showed in [HLR95] that media access strategies based on token mechanisms are not suitable for our purposes.

Of course, it might be argued that a protocol like SSCMP, which replaces small sequence-numbers by means of a long timestamp, is of questionable use in a fieldbus. However, remember that "real" real-time applications require timestamps anyway!

Since we are aiming at immediate applicability, we will build our NCSCs for widely used, standardized processor bus systems. Depending on the size of the resulting board, this will result in VME-modules or, preferably, in small-sized *M-modules*. Initiated by the companies MEN, Philips, and others, and promoted by the german MUMM e.V., the low-cost, multi-vendor M-modules piggyback I/O-system now provides dozens of different process interface modules, see [MMSa] and [MMSb] for details.

3.1.3 Theoretical Research

Apart from the theoretical analysis that needs to be done to prove the suitability of the concepts underlying the protocol's software and hardware, there are also two more or less seperate issues to be considered:

- *Protocol behaviour under failure conditions*

  It is important to explore *systematically* the performance/correctness penalties associated with failures and to conceive measures for improvement —a problem, which has been generally neglected in the research work surveyed in Section 2. For example, a question of particular importance is how long an earlier failure may affect the execution of the protocol after it has ceased to exist. Contamination and inconsistency due to atomic broadcasts initiated by faulty nodes also require special attention, cf. [GT91].

- *Formal correctness proofs*

  Designing a protocol suitable for fault-tolerant —possibly safety-critical— real-time applications makes it mandatory to devise correctness proofs. That is, certain safety and lifeness properties must be proven to hold for the specification (conceptual basis)

of the protocol. Formal methods based on assertions have successfully been applied for that purpose, see [ZH95] and [JL87] for only two examples. However, since the ancestor of our SSCMP (the SCMP of [LSW91]) has sucessfully been analyzed by means of abstraction functions in [Lam93], we will adopt this approach —actually some extension, the timed automaton model of [SLL93]— for our purpose.

Ultimately, a formal proof for the actual implementation would be required; however, it is rather questionable whether such a proof is manageable. What might possibly be done here is to apply protocol verification techniques supported by tools like [Tur93].

3.1.4 Experimental Evaluation

Experimental evaluation of the properties of the hardware and software implementation of SSCMP is of course mandatory. Apart from the fact that a formal correctness proof of the actual implementation is not likely to be provided, experiments are usually the only means to assess the performance of the implementation. We are planning a two-step approach here:

1. *Simulation*

   The implementation of the protocol should first be tested and evaluated in a suitable simulation framework. Modelling the behaviour of the underlying distributed system by means of a powerful discrete-event simulation system and performing initial implementation tests on top of it is necessesary for primarily two reasons:

   - We cannot afford to delay initial software testing until all hardware development is completed; this would intolerably impair concurrent project work.
   - Performing initial testing/evaluation of the combined hardware and software implementation would contradict the most elementary principle of testing, i.e., modularity.

2. *Experimental evaluation of NCSCs in a dedicated testbed*

   As soon as the implementation has passed the simulation phase, it is possible to test and evaluate the combination of software implementation and NCSC in a suitable testbed. Actually, we will build a distributed system comprising several CPUs equipped with an NCSC each, running a distributed testing application.

   To reduce the development work associated with testing, we will use the same testing application for all NCSCs. More specifically, we will develop this application to run under the pSOS^{+m} operating system. pSOS^{+m} is easily extended to support the NCSC in question by providing a specific device driver software; note that the detailled concept of this driver has already been developed in [Pus95].

3.2 Project Implementation

In this section, we will describe how we are going to implement the project in order to approach the goals listed in the previous section. Before presenting our actual workplan

we will briefly survey the major reasons why we think to be able to perform this project successfully.

- We consider it most encouraging that [SP95] has been accepted for publication in the leading international journal *Computer Networks and ISDN Systems*; all referees of the paper agreed that SSCMP is novel and worth to be explored further in more detail.

- We have already a quite clear idea how most issues of Section 3.1 can be attacked. After all, the results of [SP95] were obtained some time ago.

- Performing research on SSCMP may exploit several synergies with our —by now well-advanced— project SynUTC, not only by using the project's primary results but also by reusing (parts of) the testbed.

  By the time of submission, we have already completed the functional specification ([SS95]) and also the most important parts of the design ([Loy95]) of the UTCSU-ASIC. The concepts underlying the software part of SynUTC (interval-based clock validation, see [Sch95]) are resonably complete and sound; we are currently working on proofs of precision and accuracy bounds. There is also a prototype version of an NCSC-Ethernet ([Hor94]) and, most importantly, the detailled concept ([Pus95]) of a pSOS^{+m} device driver for this NCSC; the latter is currently being implemented and may be reused for the SSCMP testbed. Finally, we are about to start experimental evaluation of GPS satellite receivers.

- Earlier reseach done by the authors establishes the necessary scientific basis for the project (cf. the attached lists of publications):

  - *U. Schmid* (*Dept. of Automation*) has been working on several related problems in the field of computer communications and distributed real-time systems for several years, in particular on collision resolution in random multi-access channels and scheduling in real-time systems.
  - *W. Kastner* (*Dept. of Automation*) areas of research are distributed systems and formal verification.
  - *D. Loy* (*Dept. of Computer Technology*) has considerable expertise in ASIC design and testing, in particular in the area of fieldbusses.

- The basic conditions to perform the project in question are nearly optimal:

  - Working together on the SynUTC-project for some time, we may say that the cooperation bewteen our departments works fine. In fact, our individual capabilities and experiences are complementing one another almost optimally.
  - Exploiting the abovementioned synergies with the SynUTC-Project is efficient and cost saving.
  - The Department of Computer Technology participates in ESPRIT *Eurochip* (and will certainly participate in the following *Euro-practice* programme as well), so that developing and manufacturing an ASIC is easy and costs a small fraction of the usual fees only.

- We have good connections to the french research lab INRIA in Rocquencourt, in particular to the group of *Gerard Le Lann*, who invented *Deterministic Ethernet* and, most importantly, DOD-CSMA-CD, cf. Section 2.2. In fact, G. Le Lann has consented to cooperate with us on this project. Needless to say, this should help us considerably in dealing with the topic timely delivery of messages.

Last but not least, we have a clear picture of the how to implement the project, i.e., the following workplan:

1 **Development of SSCMP concepts**: all participants, 6 months (U. Schmid)[8]

We first need a comprehensive picture of how all the different features of SSCMP listed in Section 3.1.1 are to be provided. Although there is no need for full proofs of suitablility at this stage, we have to be reasonably certain that the concept chosen for each issue will work. In particular, we have to explore all the details underlying the different networks supported (CAN, Ethernet, cf. 3.1.2) in order to judge the suitability of a certain design decision. Note that we will begin exploring those issues prior to the official start of the project.

A major result of this first step is the functional specification of the SSCMP hardware support. This specification is the primary interface between the computer science and electrotechnical part of the project, which may then work quite independently of each other.

2.1 **Developing detailled concepts**: NN1, NN2, 1 year (U. Schmid, W. Kastner)

In this phase, detailled concepts for

- (improved) connection management including atomic multicast (NN2),
- *real-time virtual circuits* for timely delivery (NN1),

must be provided, possibly including sketches of formal proofs of correctness. If an idea initially considered appropriate is found to be unsuitable at this stage, it should still be manageable to change the design of SSCMP hardware support accordingly. Both issues listed above involve considerable research, conceptual, implementation, and (simulation) testing work and should be honoured by a full contract of employment each.

2.2 **Developing the SSCMP hardware support**: M. Horauer, 1 year (D. Loy)

Based on the functional specification, an ASIC providing the functionality outlined in Section 3.1.2 is to be designed. It seems that the required features are pretty non-standard, so that there is not much hope of being successful with off-the-shelf devices. Detailled concepts of the NCSCs (CAN, Ethernet) must also be developed during this phase.

This work obviously needs an expert in the field of hardware design. Fortunately, we have such a person at hand: *Martin Horauer* (supervised by Dietmar Loy)

[8]The supervisor responsible for coordinating/directing the appropriate phase is given in parantheses.

is currently implementing the UTCSU-ASIC and also the NCSC-Ethernet in our SynUTC-Project. The value of his experience and, in particular, the benefits of his familiarity with the issues relevant for SSCMP cannot be overstated.

2.3 **Development of evaluation testbeds**: NN3, 1.5 years (NN1)

Concurrently with 2.1 and 2.2, experimental evaluation must be planned and built, cf. Section 3.1.4. It is also necessary to define evaluation criteria and to implement hardware and software for

- simulation (in particular, the SSCMP hardware support and network controller needs to be simulated for multiple nodes),
- NCSC testbed,
- evaluation and interpretation of measurement results (should ideally be the same for simulation and testbed).

Implementation can be done by a very good student working on his/her diploma; however, (half) a research stipendium is certainly necessary to compensate for the unusual amount of work.

3.1 **Developing detailled proofs and implementation**: NN1, NN2, NN4c, NN5c, 6 months (W. Kastner, U. Schmid)

At this stage, detailled proofs of correctness must be worked out and the final implementations (for CAN, Ethernet) must be provided.

Implementation of the SSCMP protocol software will be done primarily in C++ or C, which is suitable for being used in simulation and for incorporating SSCMP in a pSOS^{+m} device driver according to the concept developed in [Pus95]. However, it is most likely that there is also some low-level (microcode) programming required for implementing MAC algorithms suitable for timely delivery of messages, cf. Section 3.1.1. This work can be performed by very good students within their diplomas, supported by (half) a research stipendium to compensate for the difficult work.

3.1 **Development of NCSC hardware**: NN4e, NN5e, 6 months (M. Horauer)

In this phase, which should reasonably overlap with the actual development of the SSCMP hardware support, the NCSCs (CAN, Ethernet) must be built. This may be done efficiently by students (NN4e, NN5e) working on diplomas, who can rely upon the specification worked out in the previous phase. However, the considerable amount of work calls for support by (half) a research stipendium.

4 **Experimental testing and evaluation in the NCSC testbed**: NN1, NN2, M. Horauer, NN3, 6 months (W. Kastner, D. Loy, U. Schmid)

In this last phase, the implementation of SSCMP must be tested and evaluated by means of the NCSC testbed; completing formal correctness proofs and (tool-supported) protocol verification may take place at this stage as well.

Note that implementing the above workplan —which does not provide any slacktime— takes 2.5 years. We will try to start the first phase of the project before the official start in order to arrive at a funded period of 2 years. However, it might well be the case that we will need some additional time for completing the work.

4 Required Support

4.1 Location

The computer science and electrotechnical part, respectively, of the project will be performed at

- *Department of Automation* (E183/1, headed by *Prof. Schildt*),
- *Department of Computer Technology* (E384, headed by *Prof. Eier*),

both at Technical University of Vienna. The departments heads' declaration of consent for using the departments' infrastructure are enclosed.

4.2 Staff

4.2.1 Available Staff

- *Univ. Doz. Dr. Ulrich Schmid* (*Dept. of Automation*)
- *Univ. Ass. DI Wolfgang Kastner* (*Dept. of Automation*)
- *Univ. Ass. DI Dietmar Loy* (*Dept. of Computer Technology*)

4.2.2 Required Staff

According to the workplan in Section 3.2, we will need the following additional staff:

- **1. year**
  - *DI Martin Horauer*: full contract of employment for 1 year
  - *DI NN1*: full contract of employment for 1 year
  - *DI NN2*: full contract of employment for 1 year
  - *NN3*: research stipendium for 1 year (ATS 5.000,-/month)
- **2. year**
  - *DI Martin Horauer*: full contract of employment for 1 year
  - *DI NN1*: full contract of employment for 1 year
  - *DI NN2*: full contract of employment for 1 year
  - *NN3*: research stipendium for 1 year (ATS 5.000,-/month)

- *NN4c*: research stipendium for 6 months (ATS 5.000,-/month)
- *NN5c*: research stipendium for 6 months (ATS 5.000,-/month)
- *NN4e*: research stipendium for 6 months (ATS 5.000,-/month)
- *NN5e*: research stipendium for 6 months (ATS 5.000,-/month)

The following table summarizes the costs:

#	Σ	1.Y	2.Y	item	ATS (incl.)
1	2	1	1	employment *DI Horauer*, ATS 282.000,-/year	564.000,-
2	2	1	1	employment *DI NN1*, ATS 282.000,-/year	564.000,-
3	2	1	1	employment *DI NN2*, ATS 282.000,-/year	564.000,-
4	2	1	1	research stipendium *NN3*, ATS 5.000,-/month	120.000,-
5	0.5	-	0.5	research stipendium *NN4c*, ATS 5.000,-/month	30.000,-
6	0.5	-	0.5	research stipendium *NN5c*, ATS 5.000,-/month	30.000,-
7	0.5	-	0.5	research stipendium *NN4e*, ATS 5.000,-/month	30.000,-
8	0.5	-	0.5	research stipendium *NN5e*, ATS 5.000,-/month	30.000,-

4.3 Equipment

4.3.1 Available Equipment

The departments' infrastructure and equipment may be used by the project to a reasonable extent. In particular, there is a powerful Sun workstation running *Cadence* ASIC design software available for the project at the Department of Computer Technology. VME-racks for mounting the VME-Modules and pSOS^{+m} operating system licenses for M680xx processors are provided by the Department of Automation. Moreover, a Sun workstation running a pSOS^{+m} software development environment for C and C++ used in the project SynUTC (P10244-ÖMA) may be (sharedly) utilized in the SSCMP-project as well.

4.3.2 Required Equipment

In order to perform the simulation described in Section 3.1.4, we need a powerful discrete-event simulation software for a Sun workstation. However, since an evaluation of the numerous existing simulation systems is a time-consuming task, we do not know by now which simulation software will suit our needs best (we estimated the resulting costs based on our experience with software of similar size).

We will also need a suitable software for evaluation and interpretation of measurement results (for an existing Sun workstation). The well-known Mathematica software, which may be obtained cheap from the campus software support, should suit our needs quite well.

Finally, there is a need for purchasing a microcode-development environment for the programmable network controller (e.g., the 68EN360) that will eventually be utilized. Again, the resulting costs have been estimated. If it should happen that we cannot utilize a programmable controller at all (due to our special requirements), we have to develop suitable VLSI network controller chips instead; manufacturing costs will replace the costs for the development environment in that case.

In order to build the testbed for experimental evaluation of the full SSCMP implementation, in particular, the NCSC M-Modules (see Section 3.1.2), we need four powerful VME CPU-Modules capable of carrying at least 2 M-Modules. The latter requirement comes from the fact that a single NCSC will likely consume two standard M-Modules in size. Each CPU should also have an ordinary Ethernet Controller on board, which is needed for connection to the software development workstation, and a few digital I/O lines for testing purposes. Four CPUs are required (at least) due to the fact that we have to cope with byzantine faults, and it is well-known that, in order to mask only a single byzantine fault, at least 4 nodes are required in the distributed system. Finally, in order to run our pSOS operating system on the CPUs, a board support software package (containing drivers for the onboard I/O devices) is required.

The following table summarizes the expected costs (1 DFL $\approx$ 6.5 ATS):

#	Σ	1.Y	2.Y	item	ATS (incl.)
9	1	1	–	Discrete-event simulation software for Sun	.80.000,–
10	1	1	–	Mathematica for Sun	7.500,–
11	1	1	–	Microcode-development software	50.000,–
12	4	1	3	VME M-Modul CPUs, à DFL 6.804,– (excl.)	212.285,–
13	1	1	–	pSOS Board Support Package, à DFL 3.500,– (excl.)	27.300,–

4.4 Required Material

Material is primarily required for building two different NCSC-boards (redundant CAN, redundant Ethernet):

- Manufacturing of (4–8 layers) multilayer printed circuit board will cost about ATS 20.000,–; however, one redesign has to be considered.
- Components and SMD assembly for 5 NCSC-boards (1 prototype+4 boards for evaluation after redesign); costs are estimated to be ATS 7.000,– per board on the average.

The following table summarizes the resulting costs:

#	Σ	1.Y	2.Y	item	ATS (incl.)
14	5	–	5	5 boards NCSC-CAN	75.000,–
15	5	–	5	5 boards NCSC-Ethernet	75.000,–

4.5 Travelling Costs

Travelling costs are required for two purposes, namely (1) presentation of project results at conferences and (2) visiting INRIA Rocquencourt (G. Le Lann's group) for cooperation, see Section 3.2. Whereas it does not make sense to us to make any plans for conferences now (we will apply for funding when the need arises), it is possible to estimate the costs arising from cooperating with INRIA.

First of all, it is certainly necessary for the project head to pay a one-week's visit to INRIA at the beginning of the project in order to negotiate actual cooperation, acquire latest results, and present our work. Second, funding is needed for travelling and lodging of employee NN1 (working on timely transmission) for a 1 month visit during the first year. Planning for the second project year is of course more difficult in advance, but we think that two visits for two weeks each should be appropriate. One APEX flight Vienna-Paris-Vienna costs ATS 5.050,–, and boarding and logding in an average hotel is FF 450,– per night. Official refunding per day is ATS 285,–. The resulting costs are summarized in the table below (1 FF $\approx$ 2 ATS).

#	Σ	1.Y	2.Y	item	ATS (incl.)
16	4	2	2	costs for visiting INRIA Rocquencourt	94.855,–

4.6 Other Costs

We need support for manufacturing a small batch (20 units) of the SSCMP-ASIC described in Section 3.1.2. The costs include:

- Chip complexity ($\Longrightarrow$ die size) is certainly high (based on the experience with the UTCSU-ASIC in the SynUTC-project, we think that 20000 gates will be appropriate). Currently, the Austrian manufacturer *AMS* offers 0.8 μm CMOS technology with 800–900 gates/mm^2, at the price of 120 ECU/mm^2 (incl. 20% VAT) for ESPRIT *Eurochip* participants (and 342,– ECU/mm^2 for commercial users). Note that the Austrian AMS is considerably more expensive than foreign manufacturers (like ES2, for example, which is however likely to leave the Eurochip business). Our requirements amount to approximately 25 mm^2 die size.
- Implementing boundary scan increases chip space by approx. 10%.
- Large PGA package required in order to provide 100+ pins, which become necessary due to the required mapping of logical sender/receiver ids (32+ bit) to physical ones (32+ bit). Additional 12.5 mm^2 of die size are required for bonding here.
- One complete redesign of the whole chip, which is inevitable when ASICs of this complexity are designed.

The following table summarizes the costs (1 ECU $\approx$ 15 ATS):

#	Σ	1.Y	2.Y	item	ATS (incl.)
17	2	1	1	manufacturing ASIC, à ATS 72.000,-	144.000,–

A Paper to appear in Computer Networks & ISDN Systems

B International Cooperation with INRIA

C Curriculum Vitae

D Official Forms, Offers