A SCALABLE SCHEDULING MECHANISM WITH APPLICATION TO AAL2/ATM MULTIPLEXING

Michael Menth
University of Würzburg, Am Hubland, D-97074 Würzburg, Germany
Phone: +49 931 8886644, E-Mail: menth@informatik.uni-wuerzburg.de

Stefan Schneeberger
Siemens AG, Hofmannstraße 51, D-81359 Munich, Germany
Phone: +49 89 72247155, E-Mail: Stefan.Schneeberger@icn.siemens.de

ABSTRACT

This paper describes the architecture of an ATM scheduler that performs cell spacing and respects additional time constraints. The scalable timing mechanism allows for real-time execution even in presence of many ATM virtual channel connections. The algorithm is suited as reference for special purpose implementations and the core idea can be adapted for scheduling problems in IP networks. In particular, a scheduling algorithm for an AAL2/ATM multiplexing device is presented.

1. Introduction

The $3rd$ Generation Partnership Project (3GPP) is the standardization body for the future Universal Mobile Telecommunication System (UMTS). UMTS will support low-bitrate real-time services like voice, circuit switched data, and packet switched data to enable multimedia conferences for wireless clients. 3GPP suggest the Asynchronous Transfer Mode (ATM) as the transmission technology for the UMTS terrestrial radio access network (UTRAN) [1, 2, 3] and its use in the core network is also discussed [4].

In the UTRAN, the traffic is often transported over expensive leased lines and the operators of mobile networks try to utilize them as efficiently as possible. ATM cells offer a fixed payload data unit (PDU) of 48 bytes. In case of compressed voice, packets have an average payload size of only 20 bytes, which yields a low bandwidth utilization of about 40%. To overcome this problem for low-bitrate data, the ATM Adaptation Layer Type 2 (AAL2) [5] is used: Several small sized packets are multiplexed into the PDU of a single ATM cell, which leads to a cell utilization of almost 100% [6, 7, 8]. An overview of AAL2/ATM switching technology is given in [9, 10]. Further performance studies for AAL2 are addressed in [11, 12].

The AAL2/ATM protocol suite is mandatory in the first and second release of UMTS and may be applied to both the access and the core network. Hence, vendors are forced to offer AAL2/ATM transmission equipment. However, the scheduling in the multiplexing devices is not trivial. Packets are delayed when they are multiplexed into an ATM cell using AAL2 and a timer limits the multiplexing interval. An ATM cell is ready for transmission if it is filled or if its timer has expired. ATM cells of the same ATM connection require a certain inter-cell distance according to the ATM traffic contract. In addition, many ATM connections compose an ATM pipe. Hence, the scheduler has to respect time constraints for AAL2 multiplexing, ATM cell spacing, and ATM multiplexing. This is depicted in Figure 1.

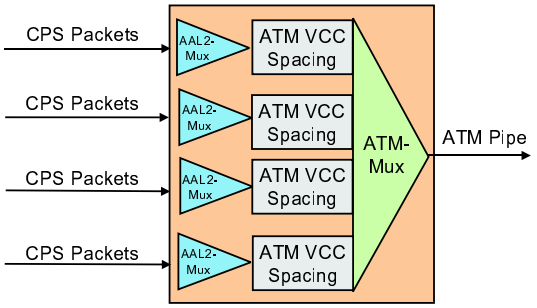

Figure 1: Functional description of an AAL2/ATM scheduler.

ATM cells are also delayed by ATM cell spacing and ATM multiplexing. We take advantage of that fact by implementing the three tasks in a single control unit. So, AAL2 multiplexing can still be performed during the ATM cell spacing time and ATM multiplexing time. This yields a better overall performance of the system [13]. The algorithm must execute within the time of a single ATM cell cycle because the scheduler runs in real-time. Moreover, the runtime of the algorithm may not exceed the duration of a cell cycle regardless of how many ATM connections

are served.

This paper gives a modular description for an ATM scheduler with a timing system such that its runtime is independent of the number of served ATM connections. Furthermore, the scheduler incorporates a distributed spacing algorithm. We call it the STRS Scheduler because its runtime scales well with the number of ATM connections, it has a timing mechanism, the timers can be reset, and ATM cell spacing can be performed. We enhance the STRS Scheduler to perform the above mentioned AAL2/ATM scheduling tasks. The concept of the STRS Scheduler can be reused for the design of other ATM and IP packet scheduling mechanisms.

The paper is organized as follows. In Section 2, ATM and AAL2 are presented and requirements for the AAL2/ATM scheduling mechanism are derived. Section 3 describes a scalable timing system, a distributed spacing algorithm, as well as the overall architecture of the STRS Scheduler. Section 4 illustrates the use of the STRS Scheduler by customizing it for AAL2/ATM multiplexing. The last section summarizes the work and gives an outlook on further extensions.

2. Low-Bitrate Real-Time Multiplexing in ATM Networks

In this section, we describe the UTRAN as a scenario where low-bitrate real-time traffic prevails. We give a short introduction to ATM and explain how AAL2 improves the utilization of the network resources. Finally, we outline the requirements for an AAL2/ATM scheduling algorithm.

2.1. UMTS Terrestrial Radio Access Network

In UMTS, a mobile handset communicates over the air interface with a base station, called NodeB, which is driven by the radio network controller (RNC). The data travel over the RNC to the UMTS mobile switching center (UMSC) and are forwarded from there either into the UMTS core network or into the public switched telephone network (PSTN). If possible, the mobile transmits data over the wireless link to up to three NodeBs, simultaneously. This is called macro-diversity and helps to improve the transmission quality in wideband CDMA systems. The data arrive via the corresponding drift-RNCs (D-RNC) at the serving-RNC (S-RNC) where upstream signals are combined, i.e., the best packet is chosen, and downstream data are multicast (Figure 2). The different legs of macro-diversity need to send data synchronously to the mobile terminal, therefore, even packet switched data have real-time constraints in the UTRAN. The bitrate of the traffic streams ranges from 7.6 kbps for voice up to 64 kbps for CS data. Hence, mostly low-bitrate real-time traffic is transported in the UTRAN, so it makes sense to employ techniques to improve the utilization of the resources in the wireline part of the network.

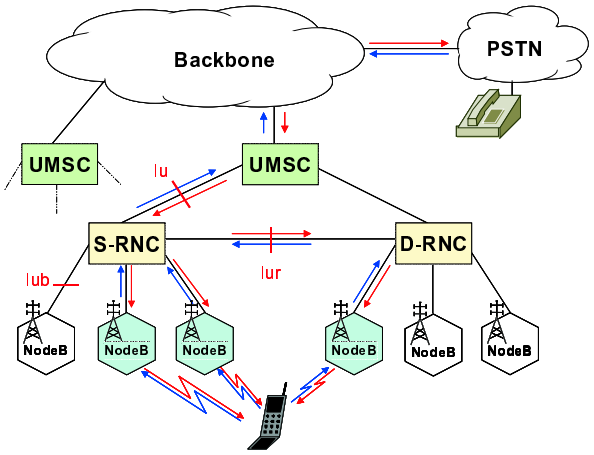

Figure 2: Macro-diversity in UMTS.

2.2. Asynchronous Transfer Mode

ATM networks are packet switched and connection oriented. The basic transportation units are cells with a fixed PDU of 48 bytes and a 5 bytes protocol header.

The connections are virtual, i.e., the bandwidth is shared among different ATM virtual channel connections (VCCs) by asynchronous multiplexing of cells that belong to different VCCs. The cells of a VCC are not assigned to a specific time slot as in time division multiplexing (TDM) systems, they are rather forwarded as they arrive and cells of different VCCs contribute to a multiplexed ATM cell stream in an ATM pipe. It is possible that more than one ATM cell compete for the common ATM outlet at a time but only one can be admitted. Therefore, an ATM cell scheduler controls the ATM outlet and assigns sending permission to the ATM VCCs and ATM multiplexing delay occurs.

For every ATM VCC, an ATM traffic contract is negotiated. It contains the desired quality of service (QoS) and a traffic descriptor. This information is needed for the bandwidth management in ATM switches to offer real-time guarantees. When an ATM VCC is connected to a different network, the ATM network policer controls the traffic characteristics of the ATM cell stream. Non-conforming cells are either immediately discarded or marked with the cell loss priority (CLP) bit and dropped later in case of network congestion. Hence, to meet the ATM traffic contract and to avoid cell losses by policing, the traffic source has to delay the ATM cells such that the traffic descriptors are met. This is called ATM cell spacing.

2.3. ATM Adaptation Layer Type 2

Data compression before transmission is indispensable for wireless applications because the wireless capacity is very expensive. So, compression engines like the the adaptive multi-rate (AMR) vocoder are employed in the mobile terminals of UMTS. Their output consists of a periodic stream of compressed voice samples of about 20

bytes. If they are carried individually in separate ATM cells, the bandwidth utilization by user data is fairly low ($\frac{20\ bytes}{53\ bytes} = 37.7\%$) due to wasted payload.

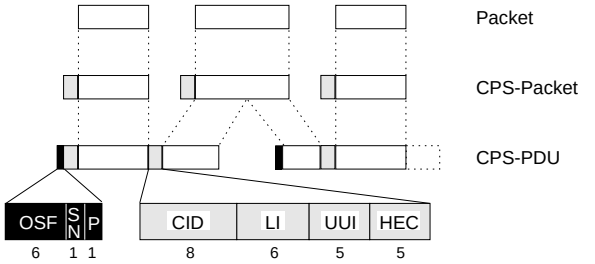

Figure 3: The AAL2 protocol.

To overcome this problem, AAL2 is conceived for multiplexing several packets into one ATM cell. This pertains to low-bitrate real-time traffic in general and is not limited to voice applications. According to Figure 3, the packets are equipped with a 3 bytes header which carries amongst others the one octet connection identifier (CID), and the length indicator (LI). User to user information (UUI) allows for inband signaling and header error control information (HEC) makes the protocol robust against bit errors. The packets including the AAL2 packet header are called common part sublayer (CPS) packets. The payload of an ATM cell used for AAL2 application begins with a one byte starter field. It carries an offset field (OSF) that indicates where the next CPS packet starts. A sequence number (SN) and a parity bit (P) protect the header information. Hence, 47 bytes are left for the common part sublayer protocol data unit (CPS-PDU) of the AAL2 protocol. The CPS packets are contiguously placed one after another into the CPS-PDU and, if necessary, CPS packet splitting over ATM cell boundaries is performed. Because of a limited CID size and some reserved values, only 248 different connections can be differentiated within a single ATM connection. They are called AAL2 connections and their ensemble constitutes the AAL2 path. This is illustrated for AAL2 trunking in Figure 4. Packets from different connections are delayed and accommodated in a single ATM cell. This AAL2 multiplexing time is limited by a timer common usage, so we denote the multiplexing time by TCU. The CID information is stored in the switches of the trunking endpoints and is used for AAL2 multiplexing and demultiplexing.

If the AAL2 connections of a single AAL2 path share only the next link, AAL2 switching is necessary. It is a combination of demultiplexing the AAL2 paths, regrouping the connections, and multiplexing them into new AAL2 paths. Such an AAL2 switching unit may be applied in UMTS network elements like RNC and UMSC or it may be even used in a switching node within an ATM/AAL2 backbone network.

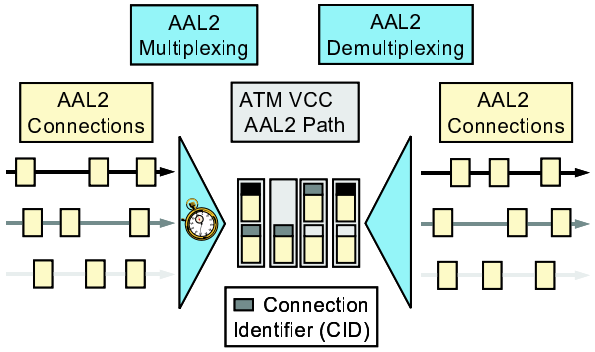

Figure 4: AAL2 trunking.

2.4. Requirements for an AAL2/ATM Scheduler

An AAL2/ATM transmission unit performs the following tasks. It multiplexes CPS packets from different AAL2 connections that belong to the same AAL2 path into a common AAL2 multiplexing queue. An ATM cell of the corresponding VCC is sent either if it is completely filled or after TCU time. In addition, the traffic descriptor for the VCC must be respected, i.e., spacing must be performed. Eventually, many VCCs fill the ATM pipe as shown in Figure 1.

The AAL2 multiplexing time induced by TCU, the ATM cell spacing time and, finally, the ATM cell multiplexing time delay the CPS packets. Traffic delay is not desired for real-time data but in this case, it is necessary for AAL2 multiplexing.

Both the spacing time and the ATM cell multiplexing time are unavoidable, however, they should also be exploited for AAL2 multiplexing [13]. Therefore, all three functions - AAL2 multiplexing, ATM cell spacing, and ATM multiplexing - should be controlled by a single AAL2/ATM scheduler.

The system time of ATM devices proceeds in discrete time units, called cell cycles, which correspond to the duration of a cell in the switching element. This entails that all operations triggered for a specific cell cycle have to be completed within that time. Thus, an indispensable requirement for the ATM cell scheduling algorithm is simplicity to achieve this goal. In particular, the runtime of the algorithm must be independent of the number of VCCs to support large switching systems.

3. The STRS Scheduler

In this section, we present the STRS Scheduler. The name of the ATM scheduler stands for runtime scalability with respect to the number of served VCCs, for a timing mechanism, and for timer resetting as well for ATM cell spacing capabilities. The scheduler is generic, i.e, it can be easily modified because it has a simple and modular structure. The algorithm relies on a scalable timing mechanism, called calendar, that manages timers in a very efficient way. We construct a distributed spacing algorithm to

compute ATM cell spacing. Finally, we describe the adaptation of the timing mechanism to ATM, the data structure of the STRS Scheduler, and the procedures it can perform.

3.1. A Scalable Timing Mechanism

Timers are used to respect time constraints in technical systems. A simple implementation is a counter that is decreased with time. However, the computing expenses for decreasing and checking counters rise linearly with the number of used timers. This is prohibitive in real-time applications with many timers because only little CPU time can be spent for that task. Therefore, we need a scalable mechanism for the timer management.

The calendar is the time reference model of the scalable timing mechanism. It is based on a discretization of time and the metaphor "day" is used for a basic time unit. We count the time, e.g., from system start and introduce an absolute date which is an integral multiple of a day. For example, the date k corresponds to time interval $[k \cdot day; (k+1) \cdot day)$. Date variables are marked underlined in the following. We define the special purpose variable $\underline{Today}$ as the present date. It is automatically increased by the system as soon a day is over. Analogously, the variable $\underline{Tomorrow} = \underline{Today} + 1$ gives the date of the next day.

The calendar consists of an array of K slots that are numbered from 0 to $K-1$. We define a many-to-one mapping from the time domain to the calendar slots using the modulo function ($slot = \underline{date} \mod K$). Hence, every date is associated with a certain calendar slot but one calendar slot maps to many dates. We define that the calendar slots correspond to the dates from $\underline{Tomorrow}$ to $\underline{Today} + K$, i.e., only the next K future dates are represented in the calendar.

Timers are associated with certain events that are recorded in calendar entries. They are registered in the calendar for date $\underline{TimeUp}$ when the timer expires. Consequently, the maximum timer value must not exceed K days. It is possible that many timers expire at the same date, therefore, we store a set of calendar entries in a specific calendar slot.

It is important that an arbitrary calendar entry can be removed from its set in a few hardware clocks. Therefore, we propose a double linked list as an implementation of the calendar entry set. This data structure is illustrated in Figure 5. The numbers in the calendar entries relate to some specific events. A calendar entry is equipped with

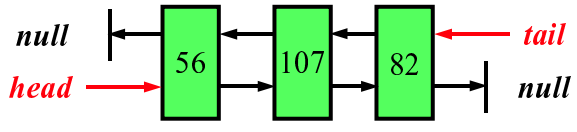

Figure 5: The double linked list of calendar entries.

a $previous$ and a $next$ pointer that recall the predecessor and the successor element in the list. A $head$ and a $tail$ pointer are handles to the beginning and the end of the list. To mark the first and the last element in the list, the corresponding $previous$ and $next$ pointer values are the terminating constant $null$. Inserting a calendar entry is almost trivial and can be performed in short time. Removing a given calendar entry from the list works as follows. The $next$ pointer of the calendar entry referred by its own $previous$ pointer has to be redirected to the calendar entry referenced by its own $next$ pointer. The same holds for the other direction. Thus, this implementation of the entry set allows for inserting a calendar entry as well as removing in a small and constant amount of time.

To set a timer, the calendar entry for the concerning event is inserted into the entry set in calendar slot $\underline{TimeUp} \mod K$ that corresponds to the expiration date $\underline{TimeUp}$ of the timer. The timer can be cancelled by removing its entry from the respective entry set. With a double linked list, it is not necessary to know the position of the entry in the calendar. A timer for an event can be reset by removing it from its set and inserting it to a new one. At the end of the present day, the set of calendar entries in slot $\underline{Tomorrow} \mod K$ is removed from the calendar. The timer for the associated events has expired and they are prepared for execution.

The strength of the calendar mechanism is that the number of operations in the calendar is independent of the number of timers. Furthermore, the insert routine executes in short and constant time because the modulo operation defines the slot where the calendar entry is inserted into the entry set. Thanks to the implementation of the entry set by a double linked list, the removing of timers from the calendar is easy and inexpensive. The restriction of this approach is the discretized time axis. However, many technical systems can tolerate that and, therefore, this timer management framework is widely applicable. It may be used for scheduling mechanisms in IP and ATM systems where many time constraints have to be respected.

3.2. A Distributed Spacing Algorithm

A network providing QoS to its customers needs to household its resources like bandwidth and buffer space and has to care that packets are not extensively delayed. To realize that, a traffic contract is negotiated where the network commits a certain QoS and in turn, the customer declares a traffic descriptor. Packets that are not conform to this declaration may be policed. The traffic characteristics are often given as a leaky or token bucket description with a certain rate and bucket depth [14, 15]. In the following, we illustrate first the token bucket on the basis of the Generic Cell Rate Algorithm ($GCRA$) that is used in ATM. Then, we conceive a distributed spacing algorithm that produces a conforming ATM cell stream. This is necessary to avoid losses due to policing.

In this section, time is again counted from system start. But now, we do not rely on integral time values but on arbitrarily accurate accounting. We mark these time variables by angle brackets. The system variable $\langle Now \rangle$ tells

the current time, i.e., the passed time since system start.

3.2.1. The Generic Cell Rate Algorithm

When an ATM VCC is established, an ATM traffic contract is negotiated. A Peak Cell Rate (PCR) and a Cell Delay Variation Tolerance ($CDVT$) are declared for the service category Constant Bit Rate (CBR). The conformance of the ATM cell stream is controlled by policers at the borders of an ATM network using the $GCRA$.

With every ATM VCC, a variable $\langle TAT \rangle$ is associated that tells the theoretical arrival time for its next ATM cell. This variable may be initialized with $-\infty$. Algorithm 1 is invoked whenever an ATM cell arrives. The arrival time of an ATM cell is given by the system time $\langle Now \rangle$. If an ATM cell arrives not sooner than $\langle TAT \rangle$ respecting some tolerance L (limit), the ATM cell is conform according to $GCRA(I, L)$, otherwise not. If the ATM cell is conform, the variable $\langle TAT \rangle$ is updated using the increment I that denotes negotiated inter-cell distance of two consecutive cells of a VCC. In IP networks, I is proportional to the respective packet length. If the ATM cell is sent earlier than $\langle TAT \rangle$, the new theoretical arrival time is $\langle TAT \rangle + I$ to log the claiming of the tolerance L. Otherwise, it is set to $\langle Now \rangle + I$.

Algorithm 1: $GCRA$.

Input: Increment I, Limit L
  if $\langle TAT \rangle - L \leq \langle Now \rangle$ **then** {cell arrives late enough}
    $Conformance := true$
    if $\langle Now \rangle < \langle TAT \rangle$ **then** {cell arrives earlier than $\langle TAT \rangle$}
      $\langle TAT \rangle := \langle now > \rangle + I$
    else {cell arrives later or equal to $\langle TAT \rangle$}
      $\langle TAT \rangle := \langle TAT \rangle + I$
    end if
  else {cell arrives too early}
    $Conformance := false$
  end if
Output: $Conformance$

3.3. A Distributed Spacing Algorithm

For service category CBR, the ATM cell stream has to comply with the negotiated PCR within a certain $CDVT$. It must be $GCRA(1/PCR, CDVT)$ conform. To avoid losses due to policing, the ATM traffic contract has to be respected when an ATM cell stream is generated. To that aim, a spacer delays the ATM cell so long that they comply with the ATM traffic contract. We present now a distributed spacing algorithm that helps generating an ATM cell stream that is $GCRA$ conform.

The algorithm relies on two functions. The function $SpacingUpdate$ is called when an ATM cell is sent and records the next theoretical transmission time ($\langle TTT \rangle$), which is analogous to $\langle TAT \rangle$ in the $GCRA$. Initially, $\langle TTT \rangle$ may be also set to $-\infty$. When a sending request is made, the function $SpacingNext$ computes the next transmission time $\langle NTT \rangle$ according to the $GCRA(L, I)$.

3.3.1. The Update Algorithm $SpacingUpdate$

The function $SpacingUpdate$, given in Algorithm 2, is called when an ATM cell is sent. So, the variable $\langle Now \rangle$ contains the sending time of the last ATM cell. In case that the ATM cell was sent earlier than the theoretical transmission time, the $\langle TTT \rangle$ is incremented by I. Otherwise, it is set to $\langle Now \rangle + I$. In case of service category CBR, the increment corresponds again to $\frac{1}{PCR}$. Setting $I = 0$ suppresses ATM cell spacing.

Algorithm 2: $SpacingUpdate$.

Input: $\emptyset$
  if $\langle Now \rangle < \langle TTT \rangle$ **then** {cell was send before $\langle TTT \rangle$}
    $\langle TTT \rangle := \langle TTT \rangle + I$
  else {cell was sent after or at $\langle TTT \rangle$}
    $\langle TTT \rangle := \langle Now \rangle + I$
  end if
Output: $\emptyset$

3.3.2. The Query Algorithm $SpacingNext$

When an ATM cell is to be sent at a desired transmission time $\langle DTT \rangle$, the ATM traffic contract has to be respected. The procedure $SpacingNext(\langle DTT \rangle)$ in Algorithm 3 returns the earliest cell cycle $\langle NTT \rangle$ when the cell is allowed to be sent.

If the cell wants to be sent earlier than the theoretical transmission time with some tolerance L, it can be only sent at time $\langle TTT \rangle - L$. Otherwise, it can be sent at the desired transmission time $\langle DTT \rangle$. In case of service category CBR, the limit L corresponds again to $CDVT$.

Algorithm 3: $SpacingNext$.

Input: Desired Transmission Time $\langle DTT \rangle$
  if $\langle DTT \rangle < \langle TTT \rangle - L$ **then** {cell is to be sent too early}
    $\langle NTT \rangle := \langle TTT \rangle - L$
  else {cell is to be sent late enough}
    $\langle NTT \rangle := \langle DTT \rangle$
  end if
Output: Next Transmission Time $\langle NTT \rangle$

Procedure $SpacingUpdate$ records the sending time of the last cell of an ATM VCC and procedure $SpacingNext$ is asked to find the next transmission time $\langle NTT \rangle$ when a certain desired transmission time $\langle DTT \rangle$ is requested.

The procedure $SpacingNext$ uses the theoretical transmission time $\langle TTT \rangle$ which is only correct if the previous cell is already sent and the function $SpacingUpdate$ has been called. If only one transmission request is active for an ATM VCC at a time, the algorithm works properly.

The concept of the distributed spacing algorithm is very modular. The pair of functions $SpacingUpdate$ and $SpacingNext$ encapsulates a spacing mechanism. The presented solution is easily adaptable to other spacing paradigms. Only minor changes are necessary to implement spacing for traffic contracts of the service category real-time Variable Bitrate (rt-VBR) in ATM or for the Guaranteed Quality of Service class in IP networks.

3.4. Architecture of the STRS Scheduler

Now, we put the pieces together to present the architecture of the STRS Scheduler. First, we adapt the timing mechanism to ATM systems, then, we describe the data structure of the scheduler, and finally, we specify its procedures.

3.4.1. Adaptation of the Timing Mechanism for ATM

The timing mechanism requires the introduction of an atomic time unit on which the discretization of time is based. All ATM cells have the same length and, therefore, the same transmission time. ATM transmitters proceed in these so called "cell cycles" because the cell cycle is the natural time base for ATM switching systems. Therefore, its granularity is also sufficient for timing purposes in the ATM scheduler. Hence, we define a "day" as a cell cycles and the "date" corresponds to a specific cell cycle after system start. At a speed of STM1 (155 Mbps), a day corresponds to an interval of 2.73 μsec. Allowing a date variable to be stored as a 64 bit word, a date is unique for $2^{64} = 1.6 \cdot 10^6$ years in real-time.

In the following, the variables $\langle TTT \rangle$, $\langle DTT \rangle$, and $\langle NTT \rangle$ are of any accuracy while $\underline{Today}$, $\underline{Tomorrow}$, and $\underline{TimeUp}$ are always integer values and meter the passed time since system start in cell cycles. In necessary, a time variable may be converted into a date variable and vice versa.

3.4.2. Data Structure of the STRS Scheduler

The asynchronous multiplexing of ATM cells from different VCC onto a common ATM pipe induces ATM multiplexing delay. The cells may be stored in the ATM VCC queue until they can be transmitted. The serving discipline is first-in-first-out (FIFO). A long VCC queue denotes a potentially long delay for ATM cells which is not tolerable in case of real-time traffic. Thus, a threshold indicates the maximum number of cells in the queue. If a cell surpasses that threshold, it is discarded.

The STRS Scheduler manages the ATM cell transmission among the VCCs. It has a calendar and every ATM VCC has a calendar entry to register the transmission time of the next cell in its queue in the calendar. This entry may contain the virtual connection identifier (VCI) of the ATM VCC. The VCI for an entry remains constant, a fact that should be exploited by a smart and memory saving hardware implementation.

Traffic shaping is needed for VCCs that will be policed in later network entities. The STRS Scheduler implements spacing which means that every ATM VCC stores its own set of spacing variables and traffic parameters (e.g., $\langle TTT \rangle$, PCR, and $CDVT$).

When the calendar entries leave the calendar, they are processed for execution, i.e., the respective VCCs are triggered to send their first ATM cell. However, only one ATM cell can be sent within a cell cycle and, therefore, the outdated calendar entries are stored in an ATM multiplexing request queue with FIFO discipline. It may also be implemented by a double linked list like the entry lists in the calendar. In the STRS Scheduler context, the entries in the calendar refer to VCCs for which the timer has not yet expired whereas the entries in the ATM multiplexing request queue point to VCCs that are ready for transmission. The components of the STRS Scheduler are depicted in Figure 6.

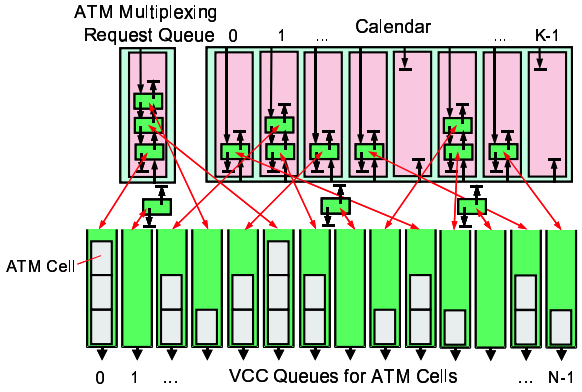

Figure 6: The components of the STRS Scheduler.

3.4.3. Registering an ATM VCC in the Calendar

We can set the timer for a VCC to send its first cell in the following way. We wish a specific transmission time $\langle DTT \rangle$ for the cell. But the minimum inter-cell distance must be respected, therefore, the next possible transmission time is determined by $\langle NTT \rangle = SpacingNext(\langle DTT \rangle)$. This time value is converted into a date variable $\underline{NTT}$ and the calendar entry of the ATM VCC is inserted into the calendar slot with the number $\underline{NTT} \mod K$. This course of actions is denoted by the function $Register(\langle DTT \rangle)$. We can cancel a timer of an ATM VCC just by removing the corresponding calendar entry from its double linked entry list in the calendar. We can change a timer to a new desired transmission time $\langle DTT \rangle$ by cancelling the old one and registering the new one.

Every ATM VCC has only one calendar entry to register a transmission request in the calendar. This assures that the distributed spacing algorithm works always correctly. There are two simple alternatives for adding a calendar entry to the entry list in a calendar slot. Inserting a calendar entry at the beginning introduces a last-in-first-out (LIFO) order within the entry list in the send queue. Appending it at the end matches the FIFO order. We consider these alternatives in the presence of spacing. An ATM VCC with a large PCR has a small inter-cell distance and its next possible transmission time is in the near future. Hence, a cell cycle in the far future will be first claimed by an ATM VCC with a small PCR. From a fairness point of view, it makes sense to serve the ATM VCCs with a high PCR earlier than those with a low PCR because they are more affected by the same absolute ATM multiplexing jitter. Therefore, we recommend appending the calendar entries at the beginning of the entry list to achieve higher priority for fast VCCs when they compete for the same cell cycle.

3.4.4. Actions at the End of a Cell Cycle

At the end of a cell cycle, the entry list in the calendar slot $\underline{Tomorrow} \mod K$ contains calendar entries that refer to ATM VCCs that are allowed to send in the next cell cycle. Therefore, this entry list is removed from its calendar slot and it is appended to the end of the ATM multiplexing request queue. The FIFO order in the ATM multiplexing request queue preserves the temporal order of the events that was given in the calendar. The time that a calendar entry passes in the ATM multiplexing request queue until transmission is exactly the ATM multiplexing delay for the associated ATM cell.

3.4.5. Actions at the Begin of a Cell Cycle

At the begin of a cell cycle, the system variables $\underline{Today}$ and $\underline{Tomorrow}$ are increased by one. If the ATM multiplexing request queue is not empty, the transmission of an ATM cell is initiated. The first calendar entry is removed from the ATM muliplexing request queue and the therein referenced ATM VCC is triggered to start the transmission of its first cell. A send flag SF is set in the cell context to avoid any read-write inconsistencies on that cell in the current cell cycle. Furthermore, the procedure $SpacingUpdate$ is invoked to update the spacing variables of the sending ATM VCC. Eventually, the ATM VCC may be again registered in the calendar if more cells are waiting for transmission.

3.5. Scheduling from the Perspective of an ATM Cell

When an ATM cell is composed, it is placed in the ATM VCC queue. When the ATM cell has reached the front position in the queue, it is registered in the calendar for transmission in cell cycle, thereby respecting the spacing constraints. When the timer expires, the complete entry list of the respective calendar slot is removed from the calendar and appended to the ATM multiplexing request queue. When the ATM VCC obtains sending permission from the ATM multiplexing request queue, the ATM cell is sent and its transmission time is recorded for the spacing of the next ATM cell in the VCC.

3.6. Runtime Considerations and Scalability Issues

We have already stated that the execution time of the routines for inserting and removing a calendar entry into or from the calendar can be achieved in constant and short time because we use a double linked list for the implementation of the entry lists. The scheduler procedures use only simple arithmetic and renounce on inpredictable recursions or loops. In particular, the runtime is independent of the number of ATM VCCs that are controlled by the scheduler. This means that the scheduler scales well for large switching systems. The overall runtime of the algorithm depends only on how often ATM VCCs are registered and how often such a request is changed.

After all, the ATM VCC queues, the calendar, and the ATM multiplexing request queue interact in a way such that spacing and ATM multiplexing can be achieved within few hardware clocks. Thus, the STRS Scheduler is an ATM cell scheduling mechanism for real-time implementations.

4. An AAL2/ATM Scheduler

An AAL2/ATM scheduler performs AAL2 multiplexing, ATM cell spacing, and ATM multiplexing. As outlined in Section 2, all three functions should be integrated in a single module to exploit the spacing and ATM multiplexing delay also for AAL2 multiplexing. The base of our AAL2/ATM scheduler is the STRS Scheduler. We define some algorithms that control the interactions between AAL2 multiplexing and the STRS Scheduler and consider again the runtime of the scheduler.

4.1. Architecture of the AAL2/ATM Scheduler

The STRS Scheduler is the base for the AAL2/ATM scheduler. The VCC queues are enhanced with some AAL2 multiplexing capabilities. The ATM cells are stored in the VCC queues during the AAL2 multiplexing interval, the ATM VCC spacing time, and ATM multiplexing time. Cell construction by AAL2 multiplexing can be performed until the ATM cell is sent, i.e., as long as the send flag SF is not set, yet. Hence, spacing and ATM multiplexing delay can be fully exploited for AAL2 multiplexing. A new ATM cell is opened for AAL2 multiplexing if the previous one is already sent, locked by the send flag SF, or already completed. The algorithms $InsertPacket$ and $SendCell$ realize the AAL2 multiplexing process and control the interaction with the STRS Scheduler.

4.2. Inserting a CPS Packet into the ATM VCC Queue

The algorithm $InsertPacket$ (Figure 7) is invoked when a CPS packet is inserted into an ATM cell of a certain VCC queue. This procedure enforces the time constraints for the AAL2 multiplexing process and interacts with the STRS Scheduler. We explain the actions that happen upon arrival of a CPS packet.

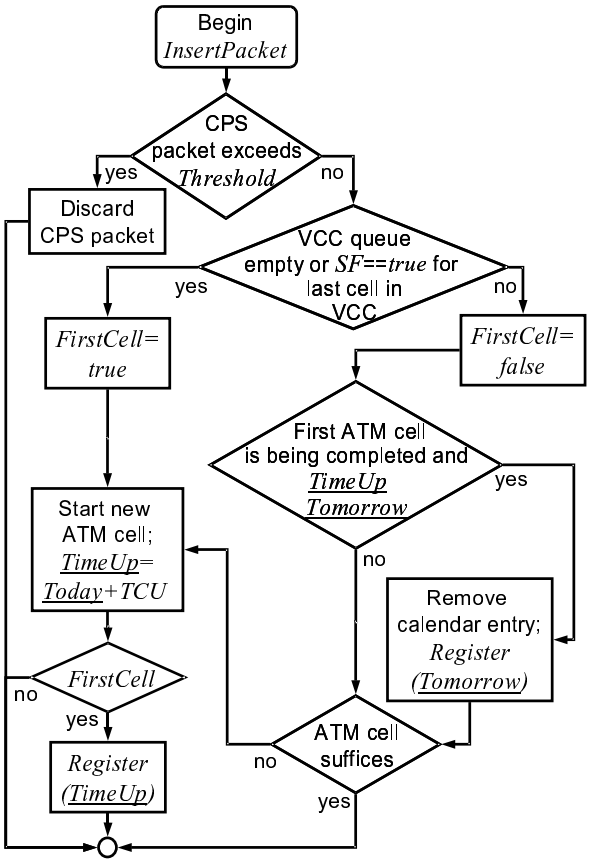

Figure 7: Flowchart of the algorithm $InsertPacket$.

If the CPS packet surpasses the VCC queue threshold, it is simply discarded. Otherwise, if the queue does not hold any other ATM cells that are not involved in a current transmission process, the CPS packet is placed into a newly constructed ATM cell. We set a timer to make the ATM cell ready for sending after TCU time even though it should not be completely filled by then. For every ATM VCC, there is a variable $\underline{TimeUp} = \underline{Today} + TCU$ that tells the expiration time of its timer. This can be done because only one ATM cell per VCC queue is being multiplexed.

We say that the VCC queue is empty if it contains no ATM cells or if the ATM cell it contains is being sent, i.e., it has its send flag SF set. If the VCC queue is empty, a new cell is started for AAL2 multiplexing and the CPS packet is accommodated into the VCC queue. In this case, the date variable $\underline{TimeUp} = \underline{Today} + TCU$ is updated. We assume that CPS packets are not larger than 46 bytes so that a single CPS packet can not completely fill a fresh ATM cell. If the ATM VCC queue was not empty, the CPS packet is placed into the last ATM cell in the VCC queue. If this is the first cell and if the CPS packet fills that cell and the AAL2 multiplexing timer has not yet expired, then the ATM VCC is registered for $\underline{Tomorrow}$ in the calendar by calling $Register(\underline{Tomorrow})$. It is possible that there is not enough space available in the ATM cell, so, a new cell is opened for AAL2 multiplexing. If the queue was empty before AAL2 multiplexing this CPS packet, the ATM VCC is registered in the calendar for time $\underline{TimeUp}$ by using the insert function $Register(\underline{TimeUp})$ of the STRS Scheduler.

The AAL2 multiplexing process for a cell is over either if the cell is completed or if its send flag SF is set. This is possible since the calendar entry is transferred after $\underline{TimeUp}$ automatically into the ATM multiplexing request queue and when it is elected for transmission, the send flag SF of the corresponding cell is set. Next, we consider the actions that are necessary upon ATM cell transmission.

4.3. Sending an ATM Cell

At the beginning of a cell cycle, the STRS Scheduler triggers the transmission of an ATM cell if the ATM multiplexing request queue is not empty. In this case, the first calendar entry of the ATM multiplexing request queue is removed and sending permission is given to the therein referenced ATM VCC. If further ATM cells are waiting in the same ATM VCC queue, the calendar entry of this ATM VCC is again inserted into the calendar. This action must be slightly enhanced for AAL2 multiplexing. In Figure 8, a flowchart for algorithm *Send-ATM-Cell* specifies this action.

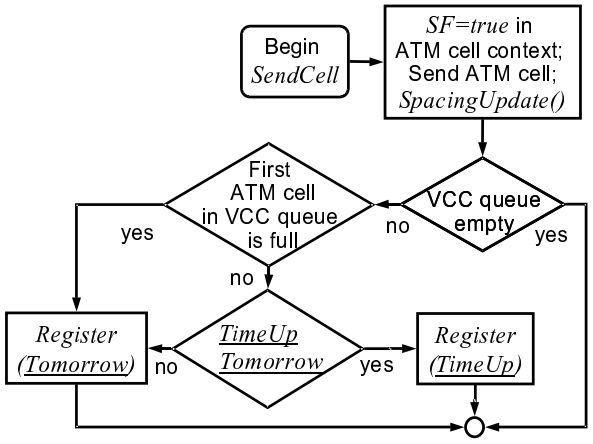

Figure 8: Flowchart of the algorithm $SendCell$.

The send flag SF in the context of the first ATM cell of the sending VCC is set. This is to avoid further write operations into this cell. The algorithm triggers the ATM VCC to send its first ATM cell and the function call $SpacingUpdate$ updates the spacing variables of the

ATM VCC. If the ATM VCC queue is now empty, no further action is taken. If there is an ATM cell in the ATM VCC queue, the algorithm differs from the STRS Scheduler. If this ATM cell is not yet completely filled with CPS packets and the timer has not yet expired, the ATM VCC is registered in the calendar for the date $\underline{TimeUp}$. Otherwise, it is registered for $\underline{Tomorrow}$. By calling the $Register$ function, the spacing conditions are automatically respected.

4.4. Runtime Considerations

We assume an AAL2 switching unit with the same input and output speed, i.e., at most one ATM cell arrives and at most one ATM cell is sent per cell cycle. In this context, we consider the runtime of the AAL2/ATM scheduler. User data have at least the size of one byte. In UTRAN applications, an additional user plane header of at least 3 bytes is added. Additional 3 bytes AAL2 CPS packet header overhead yield a minimum CPS packet size of 7 bytes. Hence, at most 7 CPS packets need to be inserted into the data structure within a cell cycle. This gives the upper bound for how often the functions $InsertPacket$ and $SendCell$ are performed. Registering the calendar entry of an ATM VCC in the calendar and determining the sending ATM VCC in the ATM multiplexing request queue are both feasible in constant time according to the architecture of the STRS Scheduler. Therefore, the presented scheduling mechanism runs for a short time that depends only on the hardware implementation of that algorithm by a linear factor. Hence, this scheduler architecture is appropriate for the real-time support of AAL2/ATM transmission technology.

5. Conclusion and Outlook

This paper proposes the algorithm for the STRS Scheduler which performs ATM cell multiplexing. Its runtime scales well with the number of served ATM VCCs. It possesses a timing mechanism that allows for resetting the timers. Its ATM cell spacing capability relies on a modular and distributed approach which makes it easy to substitute the current spacing algorithm by a different one.

Especially in access networks like the UTRAN, the AAL2 is used to achieve an efficient utilization of the link bandwidth in presence of low-bitrate real-time traffic. We described an interworking of the ATM and the AAL2 layer in an AAL2/ATM scheduler. The three required functions AAL2 muliplexing, ATM cell spacing, and ATM cell multiplexing are realized in a single device. The advantage of the presented architecture over a naive approach is that ATM cell spacing, and ATM multiplexing delay are also used for the AAL2 multiplexing process. The AAL2/ATM scheduler is based on the STRS Scheduler and can, therefore, serve several (some hundreds) AAL2 paths in real-time. Furthermore, the algorithm is easy to realize in hardware.

We conclude that the STRS Scheduler is a reference architecture for other ATM appliances. In future studies, it can be extended to priority scheduling and spacing on ATM virtual path connection basis while maintaining its runtime scalability. The scalable timer management mechanism may even be reused in IP or other technologies, e.g., for the realization of RTP/UDP/IP multiplexing [16, 17].

REFERENCES

[1] 3GPP, "3G TS25.410 version 3.3.0: UTRAN Iu Interface: General Aspects and Principles (Release 1999)," Dec. 2000.

[2] 3GPP, "3G TS25.420 version 3.2.0: UTRAN Iur Interface General Aspects and Principles (Release 1999)," Sep. 2000.

[3] 3GPP, "3G TS25.430 version 3.4.0: UTRAN Iub Interface: General Aspects and Principles (Release 1999)," Dec. 2000.

[4] ITU-T, "I.366.2 Draft Recommendation: AAL Type 2 Service Specific Convergence Sublayer for Narrow-Band Services," June 2000.

[5] ITU-T, "I.366.1 Segmentation and Reassembly: Service Specific Convergence Sublayer for the AAL Type 2," June 1998.

[6] N. Gerlich and M. Ritter, "Carrying CDMA traffic over ATM using AAL–2: A Performance Study," Technical Report, No. 188, University of Würzburg, Institute of Computer Science, Sep. 1997.

[7] N. Gerlich and M. Menth, "The Performance of AAL-2 Carrying CDMA Voice Traffic," in *11th ITC Specialist Seminar*, (Yokohama, Japan), Oct. 1998.

[8] M. Menth and N. Gerlich, "A Numerical Framework for Solving Discrete Finite Markov Models Applied to the AAL-2 Protocol," in *MMB '99, 10th GI/ITG Special Interest Conference*, (Trier), pp. 0163–0172, Sep. 1999.

[9] G. Eneroth, G. Fodor, G. Leijonhufvud, A. Racz, and I. Szabo, "Applying ATM/AAL2 as a Switching Technology in $3rd$ Generation Mobile Networks," *IEEE Communication Magazine*, vol. 37, June 1999.

[10] J. H. Baldwin, B. H. Bharucha, B. T. Doshi, S. Dravida, and S. Nanda, "AAL2 - A New ATM Adaptation Layer for Small Packet Encapsulation and Multiplexing," in *Bell Labs Technical Journal*, Spring 1999.

[11] B. Subbiah, S. Dixit, and N. R. Center, "Low-Bit-Rate Voice and Telephony over ATM in Cellular/Mobile Networks," *IEEE Personal Communications*, pp. 37–43, Dec 1999.

[12] C. Liu, S. Munir, and R. Jain, “Packing Density of Voice Trunking using AAL2,” in *Globecom*, 1999.

[13] O. Isnard, J.-M. Calmel, A.-L. Beylot, and G. Pujolle, “Performance Evaluation of AAL2 Protocol in UMTS Terrestrial Radio Access Network,” in *12th ITC Specialist Seminar*, (Lillehammer, Norway), pp. 289–300, March 2000.

[14] ITU-T, “I.371 Traffic Control and Congestion Control in B-ISDN,” March 2000.

[15] S. Shenker, C. Partridge, and R. Guerin, “RFC2212: Specification of Guaranteed Quality of Service.” ftp://ftp.isi.edu/in-notes/rfc2212.txt, Sep. 1997.

[16] M. Menth, “Carrying Wireless Traffic in UMTS over IP Using Realtime Transfer Protocol Multiplexing,” in *12th ITC Specialist Seminar*, (Lillehammer, Norway), pp. 13 – 25, March 2000.

[17] M. Menth, “The Performance of Multiplexing Voice and Circuit Switched Data in UMTS over IP Networks,” in *Protocols for Multimedia Systems (PROMS2000)*, (Cracow, Poland), pp. 312 – 321, Oct. 2000.