Title
Adaptive Baud Protocol For Wireless Communication

Authors
Eskafi, F. H.
Nassiri-toussi, K.
Liu, G.

Publication Date
1998-08-01

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

Adaptive Baud Protocol for Wireless Communication

F.H. Eskafi, K. Nassiri-Toussi, G. Liu
University of California, Berkeley

California PATH Research Report
UCB-ITS-PRR-98-27

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration.

Report for MOU **86-2**

August 1998

ISSN 1055-1425

Adaptive Baud Protocol for Wireless Communication

F. H. Eskafi[1], K. Nassiri-Toussi[1], G. Liu[1]

July 17, 1998

Abstract

We are proposing a distributed algorithm to increase the transmission rate whenever possible and decrease the rate when too much error is encountered for a system with tight real time constraint and communicating elements so that the tolerance of the system to the message losses is very limited. The transmission bauds are pre-quantized and is known to both transmitter and receiver. Due to the unobservable nature of the transmission process from the point of view of the receiver or the transmitter, there may be an inevitable asynchrony between the transmission baud and the reception baud. However, the baud change protocol is capable of resolving the asynchrony between the transmitter and receiver.

1 Introduction

Choosing the right transmission speed for a noisy environment is a challenge not yet fully answered. The transmission speed is usually been set at the initialization procedure and cannot be changed during the transmission. If the noise level increases, the data transmission is usually turned off, and the initialization procedure is restarted.

The above procedure is not acceptable for the safety critical systems which may also have time constraint for message delivery. In these instances usually redundancies are added to the communication system, i.e., a back up system is activated as soon as the primary system cannot function.

In this paper we are proposing an algorithm that increases and decreases the transmission speed according to a user-specified function. This function can be noise level of the channel, average number of packet losses, etc.. This work was motivated by a recent communication project in PATH (Partners for Advance Transit and Highway). One of the main AHS proposals that has put forth by PATH is to organize the traffic into platoons in order to increase the throughput and safety. A platoon is a group of vehicles traveling at close proximity with each other **(2-4** meters spacing). The distance between the platoons is large and depends on the velocity of the platoon, road condition, weather, etc.. Since the vehicles are automatically driven, there should be infra-structure support for routing and coordination purposes. Also Because of close spacing within the vehicles in a platoon, the vehicle controllers require information regarding the acceleration and velocity of the first car and the vehicle immediately in front. These information cannot be reliably acquired through sensors and has to be communicated. For further information regarding the structure of the proposed AHS and the controllers see [1, 2, 3, 4, 5].

[1]Department of Electrical Engineering and Computer Science, University of California, Berkeley, Ca. 94720

One proposal is to use infra-red technology for the communication between vehicles within a platoon. Infra-red provides a reliable point-to-point link capable of transmitting in different bauds.

The data transmission between vehicles takes place at fixed AT (at present it is 20 msec) intervals through a half-duplex infra-red transmission link. To ensure safety, these data transmissions must take place almost uninterrupted. Considering the fact that one wishes to have the most efficient transmission that is feasible under the circumstances, a protocol for baud regulation is required. This protocol must determine when and how a request to change the baud is initiated and how the system should proceed in every possible case, such that the maximum efficiency is achieved, while at the same time the possibility of the data transmission being interrupted is minimized. Since, we assume that all the control information data transmissions are one-sided; from the lead car *(Transmitter)* to the followers *(Receiver),* it is only logical that the transmitter should initiate any request for baud change.

The protocol will of course depend on the choice of data transmission protocol, which in this case is the Alternative-Bit Protocol (ABP). Under ABP,

1. the transmitter waits to receive an acknowledgement (ACK) from the receiver, implying that the message has been received, before it transmits the next message,
2. it retransmits the message if no ACK has been received after waiting for a certain period of time, with the number of retransmissions being possibly limited by some number, and
3. distinguishes the next message from the retransmission of the first message by adding a 1-bit flag.

Let t_0 be the amount of time the transmitter waits for an ACK before retransmitting the message. If we assume that $t_0 \geq \bar{\delta}$, where $\bar{\delta}$ is the sum of the message/ACK transmission times, the message/ACK propagation times through the communication channel and the receiver delay, in the worst case, then it is clear that there is no need to distinguish between ACKs of different messages. This protocol is clearly inefficient with respect to the wasted idle time, but this is of no importance in our case since the control data transmissions occur only at AT intervals with $AT \gg \bar{\delta}$. The types of messages in this case include the control information data packets, the test packets, and the requests for baud change packets. Also the third feature of the ABP can be ignored in this case, if the message contains updated information at every transmission time.

To design the baud change protocol, we use the alternative-bit protocol with the constraint that number of retransmissions does not exceed a certain number. Our objective is to minimize the possibility of the asynchrony between transmitter and receiver.

This paper is organized as follows. In section 2 we will explain the protocol design using a heuristic approach. In section 3 the verification procedure of the protocol is discussed. We have simulated the protocol using CSIM [6]; the simulation is described in section 4. We close the paper with conclusion in section 5.

2 Protocol Design

Our first step is to give an exact definition of the problem. As explained in the last section, certain data packets are to be transmitted from the transmitter (the leading car) to the receiver (the following car) at every AT interval. For this transmission, we use the ABP with the number of retransmissions (or timeouts) limited by N_{dt1}, where N_{dt1} is such that even if the

transmitter has to retransmit N_{dt1} times, the required time $N_{dt1}t_0 \leq \frac{1}{2}\Delta T$ at every baud level. Notice that *to*, length of the waiting period before each retransmission, depends on the baud. In particular, t_0 is chosen greater than the estimated worst-case delay between sending a message and receiving an ACK, which depends on the rate of transmissions.' The probability of failure is decreased almost exponentially as the number of retransmissions is increased. If the probability of failure for each transmission through the infra-red channel is a, then the probability that the transmitter will not receive any acknowledgement to its message is $\alpha + (1-\alpha)\alpha \approx 2\alpha$ for $a \ll 1$, while if the message is retransmitted n times, the probability that all the transmissions of the message are lost is reduced to α^n and the probability that the transmitter does not receive any ACKs is reduced to

$$\alpha^{n-1}(1-\alpha^{n+1}) \approx (n+1)\alpha^n \text{ for} \alpha \ll 1.$$

The number of retransmissions (or timeouts) which is needed to send a message successfully to the receiver, i.e., an ACK is received, is averaged over a number of AT intervals.

In each interval, after the control data has been successfully transmitted to the receiver, the transmitter decides based on this average whether it should initiate a request for baud regulation. We assume that the transmission baud is chosen from a number of pre-specified, logarithmically spaced baud levels. If the average is less than some specified low-threshold, then it is possible that a higher baud level is still acceptable, while an average larger than a high-threshold indicates that the baud should be decreased.

We further assume that at each request for baud regulation, baud may be changed by only one level and that there is no more than one request per AT interval.

Based upon the decision to regulate baud, a request for baud change, which can be repeated for at most N_{rq1} times, is sent to the receiver. After receiving an ACK for the request, the transmitter and the receiver set their bauds at the new level. At this stage, it is only logical to use the idle time to test the new baud. This can reduce the probability that control data transmissions are interrupted in case the new baud is not suitable. The number of retransmissions of the test packet is limited by N_{rq2}, where N_{rq1} and N_{rq2} are such that the total time dedicated to the baud change request/test procedure, $N_{rq1}t_0 + N_{rq2}t'_0 \leq \frac{1}{2}\Delta T$ at every baud level. (t'_0 is the waiting time before each retransmission corresponding to the new baud.)

One uses the ABP again to transmit the test packets. However, this is not possible for the baud change requests, unless receiver is capable of receiving messages with two different bauds simultaneously. Since the system is partially observed, the receiver cannot distinguish between the two situations; *1)* it has received the request and its ACK has been received by the transmitter, and 2) it has received the request but its ACK has been lost. Based on the ABP, the transmitter should change the baud and send the test packet at the new baud in the first case, while in the latter, it retransmits the request using the original baud.

Since it is not reasonable to require a hardware capable of receiving messages with different bauds at the same time, the transmitter and the receiver must be synchronized during the request period. This can be done if a version number is sent with each retransmission of the request. The transmitter sends N_{rq1} requests, each accompanied with a version number, regardless of any ACK it may receive for the requests. The version number indicates the number of additional requests that the transmitter will send. Therefore, after the first request has been received, the receiver has an estimate of the time when the last request will be sent (assuming that it knows the value of *to*, the transmitter waiting time before each retransmission.)

[1] t_0 decreases superlinearly (almost geometrically) as baud is increased level by level.

After the receiver has acknowledged the first request, provided that the channel is full-duplex, the receiver can repeat sending the ACK to the transmitter in the remaining portion of the request period regardless of any retransmission of the message it may receive. If the request has been successful, i.e., an ACK has been received by the transmitter, both bauds are changed to the new level after the request period. Next, the transmitter begins to transmit test packets at the new baud, while the receiver is also "listening" at the same level. This request protocol is clearly inefficient in terms of time conservation, but is apparently the optimal method in reducing the probability of a request failure. [2]

The test packet is transmitted, using the ABP, for at most N_{rq2} times, with the required time being equal to $N_{rq2}\, t_0$. Since the transmitter and the receiver were synchronized during the request period, the receiver has an exact estimate of the time the last test packet will be sent. If by this time no test packets or none of their ACKs have been received, then one must assume that the request has failed.

Now, ignoring the individual request or test packet transmissions, the outcome of the baud change procedure is one of the following 5 cases:

1. All the requests are lost and hence, the transmitter does not receive any ACK. No test packet is transmitted.

2. The receiver receives at least one request but the transmitter does not receive any ACK. No test packet is transmitted.

3. The transmitter and the receiver receive at least one request and one request ACK, respectively. All the test packets are lost and therefore, none of them are acknowledged.

4. The transmitter and the receiver receive at least one request and one request ACK, respectively. At least one test packet is received but the transmitter does not receive any ACK for the test packets.

5. The transmitter and the receiver receive at least one request and one request ACK, one test packet and one test-packet ACK, respectively.

Clearly, the transmitter makes the same observations in cases (1) and (2) (or (3) and (4)) and thus, can only make the same decision in these cases. Similarly, the receiver does not distinguish between case (1) and ε = (no request) (M(all requests lost) = $M(\epsilon)$), or cases (2) and (3), or cases (4) and (5). For the moment, let us consider only the baud increase requests.

Case (5) is when the request is completely successful and thus, both the transmitter and the receiver increase their baud one level. In case (1), the receiver is idle and the logical decision for the transmitter is to ignore its request for baud increase. But then, this has to be also the transmitter decision in case (2). On the other hand, since it has received a request, in this case the receiver increases its baud after by its estimate, the request period has ended. Since the objective is to have equal transmitter and receiver bauds at the end of this process and furthermore, as far as the receiver is concerned, the request has failed (no test packet is received), receiver baud is changed back to the original level after waiting for another $N_{rq2}t_0'$.

In case (3), the transmitter and the receiver both increase their bauds but after they have waited for $N_{rq2}t_0'$, the bauds are decreased to their original level. In case (4), transmitter

[2] As another option, the receiver changes its baud after the first request has been received and the transmitter begins test packet transmissions after the first ACK has been received. In spite of simplicity, the disadvantage of this method is that the probability that the transmitter receives no ACK and thus, the request fails is about α. That is to say in this case, one might as well take $N_{rq1} = 1$.

makes the same decision, while for the receiver the request has been successful and hence, its baud remains increased, which is regarded as an error! This is the only case when the two bauds are set at different levels at the end of the procedure and the probability of this event is about $N_{rq2}\alpha^{N_{rq2}}$.

One may examine all possible policies and it is easy to conclude that every other policy results in at least one case of discrepancy, with the above policy being optimum with respect to our objective.

The above protocol is represented by the following FSMs (Figure 1).

Here, AT CHECK denotes the state at which the transmitter monitors the average number of retransmissions (or timeouts) that is required in order to receive an ACK. In case a low (resp. high) number of timeouts, denoted by **Low-TO (High-TO),** is observed, a request for baud increase (decrease) is transmitted. Otherwise, the transmitter remains idle. TRANS-CH and REC_CH in Figure 1 and elsewhere refer to the FSMs representing the transmitter and the receiver transmission channels, respectively.

Note that in these machines, the part of the finite automata that represents the request period of the request/test process, has been replaced with its "untimed" or "instantaneous" equivalent. Since during this period, the transmitter and the receiver are exactly synchronized, the derivation of the untimed equivalent is rather clear and straightforward. Also, since we have assumed that N_{dt1}, N_{rq1} and N_{rq2} are chosen (with respect to ***AT*** and current bauds) such that the next sampling time never overlaps with the request/test period, it is justified to substitute this untimed process in the overall timed automata.

In addition, we should mention that in these FSMs, we do not explicitly model the timing involved in each retransmission of a message, which does not affect the analysis or verification of the FSMs. One can simply assume that the transition from WAIT PACKET ACK to SEND TEST/CONTROL PACKET has a t_0 delay.

Furthermore, implicit to all our discussions are the transmitter and the receiver channel finite state machines which essentially create a transmission delay for each transmission. These FSMs are shown in Figure 2 with TRANS and REC referring to the transmitter and the receiver FSM, respectively.

As to the requests for baud decrease, one can use exactly the same protocol. However, there is a difference in this case; if the request transmission fails (case (1) or (2)), it is an indicator that the original baud is too high, and therefore, it does not seem logical to return to or remain in that baud. In particular, a plausible scenario in this case is to decrease receiver baud if at least one request has been received regardless of the test results, and to decrease transmitter baud regardless of the request ACKs. If this procedure is followed, then the test is clearly redundant.

Note that for a baud decrease, the test determines whether the decreased baud is suitable or whether it should be decreased even more. On the other hand for a baud increase, it determines whether baud has been appropriately increased or that a wrong decision has been made.

Following the above argument, the resulting FSM for the baud decrease procedure turns out to be rather simple (although one still has to synchronize the receiver with the transmitter during the request period). There are only **3** cases: 1) When at least one request and one ACK are received and consequently both bauds are decreased. 2) When a request is received but all the ACKs are lost, where again the two bauds are decreased. ***3)*** And finally, when all the requests are lost, which results in a decreased transmitter and an unchanged receiver baud, an error in other words. The probability of the last case is only $\alpha^{N_{rq1}}$. The corresponding FSMs are shown in Figure **3,** where again the untimed equivalent of the request process has been

used.

Another advantage of this protocol design is that if the request results in error, then whether it is a request to increase or a request to decrease the baud, receiver baud will always be a level higher than transmitter baud, a property which further simplifies verification of our protocol.

As explained, in our proposed protocol, baud change requests may result in different transmitter and receiver bauds at the data transmission time,[3] therefore, we need to add a procedure to the protocol in order to correct these errors. Unfortunately, this may result in further discrepancies between the two bauds and thus, the protocol should guarantee that a few data transmissions, at least, take place when bauds are equal. To do this we introduce the notion of *base baud*

- Base bauds are the predetermined baud levels *to* which the bauds are changed when a discrepancy between transmitter and receiver bauds is observed.

To ensure that such baud is appropriate for the current channel capacity, we choose a base baud level below the current baud. In general, there are two ways to determine these baud levels: (Note that we use the same approach for both the receiver and the transmitter, but their bauds are regulated separately.)

1. Fixed: Certain baud levels $\{b_i\}$ are regarded as the base bauds with b_i the base baud for every baud R such that $b_i < R \leq b_{i+1}$.

2. Relative: the base baud is κ levels below the current level, where κ is changed whenever the baud changes:

   - $\kappa \rightarrow \kappa + 1$ for every one-level increase of the baud,
   - $\kappa \rightarrow \kappa - 1$ for every one-level decrease of the baud, and
   - $\kappa \rightarrow \kappa_0$ for every κ-level decrease of the baud, i.e., when baud is changed to the corresponding base baud, with $\kappa_0 > 2$ the initial value of κ.

This algorithm ensures that when transmitter and receiver bauds are equal, the corresponding base bauds are also equal, and when a request to change the baud results in different transmitter and receiver bauds, the corresponding base bauds are nevertheless the same.

Although both approaches are applicable, it turns out that the relative approach results in a simpler and more efficient protocol. (Consider the case when the transmitter baud $R_t = b_i$ and the receiver baud $R_r = b_i + 1$.) This is also the approach that we chose for our design.

A much simpler solution also exists where only one fixed baud level is selected as the base baud, to which transmitter and receiver bauds are changed each time a discrepancy might exist. This level can be the baud at which the transmitter and the receiver are initialized, or the lowest baud level. The problem with this method is that the selected baud level may be too high or too low with respect to the current channel capacity, consequently resulting in a less efficient solution. The protocol we have proposed, based on the more complex choice of relative base bauds, is also valid when this simpler method is applied.

Thus, if there is discrepancy between the two bauds at the data transmission period, the bauds are changed to their corresponding base bauds according to the above method. Unfortunately, the difference in bauds is not directly observable and the only indication of difference

[3] This is also the case for every other protocol.

that the transmitter (resp. the receiver) can observe is that no data transmission has been acknowledged (no data transmission has been received).

Clearly, these situations may also have been caused by sequences of selections for which the bauds are equal. Nevertheless, apparently no better solution exists, and hence, we proceeded with this design. Under this protocol, if after N_{dt1} data transmissions, the transmitter does not receive any ACK (which happens if bauds are different), transmitter baud is decreased to the next lower base baud (κ level below). The transmitter then continues to send the data at the new baud for at most another N_{dt2} times, where N_{dt2} is such that in the worst case, the time spent $N_{dt2}\, t_0 \leq \frac{1}{2}\Delta T$.

As to the receiver, let us assume, for the sake of design, that it is synchronized with the transmitter; this simplifies our analysis to a great extent. Nonetheless, we must add that the verification results indicate that as long as the receiver resets its clock with every reception of a control data transmission, the same protocol design is acceptable even when the transmitter and the receiver are asynchronous.

Thus, receiver baud is also changed to the corresponding base baud when after waiting for $N_{dt1} t_0$ from the beginning of the interval, it has not received any control data transmission (which happens if bauds are different). The receiver waits for another $N_{dt2} t_0$ and if it does not received any data again, the control data for this interval is assumed lost. Of course, at this stage the receiver may sound an alarm through a different communication link which notifies the lead-vehicle of the disconnection problem.

There are 5 possibilities which may occur in this regard:

1. Transmitter baud is one level lower than receiver baud due to an unsuccessful attempt to increase (or decrease) the baud. No communication is possible and hence, according to the protocol, both bauds are decreased to the corresponding base levels, which are equal if the relative base baud (or single base baud) approach is used. Note that since the receiver is set at a higher baud, it can have an earlier timeout. In any case, bauds will eventually be equal.

2. Transmitter and receiver bauds are equal at the beginning of the AT interval but since all the data transmissions are lost, both bauds are decreased to the same base baud. We should add that this does not indicate an inefficiency of the protocol; loss of all the data transmissions implies that the channel capacity is less than expected.

3. Transmitter and receiver bauds are equal at level R and at least one data transmission is received, but all the ACKs are lost. This situation results in error; transmitter baud changes to the corresponding base level while receiver baud is unchanged. Nonetheless, no control data has been lost in the current AT interval. The probability of this situation is about $N_{dt1}\, \alpha^{N_{dt1}}$.

   As a result, in the next AT interval, we have the following situation:

4. The transmitter is at the base baud level B, while the receiver is at the original level $R = B + \kappa_0$. No communication is possible and hence, receiver baud is decreased to the base level B while transmitter baud is decreased to the next base baud $B - \kappa_0$. If the waiting times are different, there can be *a* period during data transmissions when bauds are equal.

   Considering an asynchronous transmitter and receiver in general, we can show that to achieve this, one must have $N_{dt1}(B)\, t_0(B) \geq (2\, N_{dt1}(R) - 1)\, t_0(R) + t_0(B)$. Note that as mentioned before N_{dt1} and t_0 depend on the baud level. Since t_0 is not a design

parameter (it is chosen greater than the worst-case delay), this asks for an appropriate design of N_{dt1} based on the current baud level and the baud level at the beginning of the previous AT interval. This method seems rather complicated, but is substantially simplified when the single base baud approach is used.

5. In spite of the above design, this situation can still result in a similar situation in the next AT interval if all data transmissions or ACKs that are sent during the period the bauds are equal, are lost. The probability of this event is about $N_{dt1}(R)\,\alpha^{N_{dt1}(R)} \times (n+1)\alpha^{n}$ where $1 \leq n < N_{dt1}(B)$ depends on the choice of $N_{dt1}(B)$. This is equivalent to saying that there is a possibility that transmitter and receiver bauds will race down to the lowest baud level.

6. The data transmission is successful, therefore, bauds remain unchanged.

The FSMs corresponding to this protocol are shown in Figure 4. Together with Figures 1 and 3, these construct a FSM representing the complete transmitter/receiver system.

There are two additional facts which should be mentioned: *1)* To avoid further discrepancy between the bauds, *the transmitter initiates a request to change the baud only if the data transmission has been successful (an ACK has been received),* hence ensuring that the bauds are already equal, *and sufficient time for the request/test procedure is left in the current* AT *interval. 2)* We have assumed that bauds can be increased or decreased an unlimited number of levels. In practice, there are naturally lower and upper bounds $\bar{R}$, $\underline{R}$. This does not cause any loss of generality, since one can simply regard any baud $R \leq \underline{R}$ as equal to $\underline{R}$ and any baud $R \geq \bar{R}$ as equal to $\bar{R}$. But the following must be added to the protocol to ensure its validity: when the bauds are already at the lowest (resp. highest) level, the output of the average timeout machine AT CHECK should not indicate **Low-TO** (High_TO), a lower (higher) than normal number of timeouts, which results in a request to decrease (increase) the baud.

Our design at this stage is complete and we proceed to the protocol verification in the next section.

3 Protocol Verification

In this section, we present the protocol verification results. As noticed in the last section, the validity of the proposed protocol, specially when the receiver is not synchronized with the transmitter, is highly dependent on the timing of the requests and data transmissions. Hence, it is imperative to verify the validity of the protocol under its various timing constraints. The general approach to this problem is briefly explained in the following:

I.

Find the minimum-state untimed version of the finite automata representing the discrete event system (the protocol). [7]

Let the system involve C timers. Then at each point in time, the timer state $x \in \mathcal{R}_{+}^{C}$ represents the state of all the timers. Let $q \in Q$ be any state of the finite automata, then (q,x) is the complete state or a "snapshot" of the machine at some point in time.

We know that if the rates of increase of all the timers are equal and constant and the timing constraints involve a finite number of bounded rational numbers and are of the general form

$x_i \geq (>) x_j + k$, then $\mathcal{R}^{\mathcal{C}}_{+}$ is divided into equivalence regions $\{E;\}$such that for every $E;$ and for every q E Q, the possible transitions (edges) from (q, x) are the same for all $x \in E;$.

It can be shown that a finite number of constraints result in a finite number of equivalence regions. Therefore, one can replace all the timers with a finite state timing process whose states are the equivalence regions $\{E_i\}$ and whose transitions depend on the selections, transitions and the timing constraints of the original automata. The resulting finite automata is the untimed equivalent of the timed automata. Informally speaking one can find a finite number of snapshots which completely describe the original automata in real-time.

As discussed in [1], let the gcd of all the numbers involved in the timing constraints and their bounds be well-defined and fixed (for example all are integers less than n), then one can find a minimum decomposition of the timer space into equivalence regions which is valid for every set of above-mentioned constraints that has the same gcd and bounds. Although the number of regions obtained in this way is finite, but it can be much larger than the minimum number of regions required for a specific case.

11.

The minimization of the timing states (equivalence regions) can be done by the method of successive approximations [8].

Let the verification objective (task) be to verify $L(M) \subseteq K$, where $L(M)$ is the language generated by the automata M and K is the "desirable" prefix-closed sublanguage. We start with M_0 the original automata with the timing constraints ignored.

If $\mathcal{L}(M_0) \subseteq K$, then $\mathcal{L}(M) \subseteq K$, as well. If not, we find a selection sequence $s \in \mathcal{L}(M_0)$ such that $s \notin K$, then if s E $\mathcal{L}(M)$, it implies that $\mathcal{L}(M) \not\subseteq K$. Otherwise, there must be a set of timing constraints that are inconsistent with s (note that $s \in \mathcal{L}(M_0)$). Then, by including only these constraints in the original automata, one arrives at M_1 and the same procedure is repeated.

To verify M_k, one finds the corresponding equivalence regions in the timer space. However, it is often less tedious to do so for M_k rather than for M. One also requires an algorithm to verify $s \notin K$ and determine the constraints that it violates. An algorithm based on this method has been introduced in [2] and has been implemented in the timed version of the COSPAN software package.

Unfortunately, since in our case, the transmitter and the receiver are in general asynchronous, the minimum number of equivalence regions $\{E_i\}$ is not far below the worst case given by [7]. Nonetheless, the above iterative approach can still be a very efficient way to verify the proposed protocol. Unfortunately, due to lack of access to the timed version of COSPAN and shortage of time, we have not been able to accomplish this so far.

However, notice that none of the timing constraints in our system is expressed in terms of strict inequalities. As been conjectured (proved?), in such cases the discrete-time version of the finite automata is equivalent to the real-time automata. This is equivalent to saying that snapshots of the system at the sampling times completely describe the real-time automata.

In the discrete-time version, we assume that the timers proceed in St steps, where St, the sampling time, is the gcd of all the numbers involved in the constraints (for example 1msec in our case). This means that we need to observe the timer outputs only at the sampling times, and since all the numbers encountered in the timing constraints are bounded, this results in a finite number of timing states. Thus, in addition to the FSMs given in Figures 1, 3, 4, and 2, we also require FSMs representing the TO, RO, $T1$, and $R1$ timers. The states (and outputs) of these FSMs are integers 0 to n, where n St is the maximum output required by that timer.

Note that many of the transitions in the FSMs 1, 3, 4, and 2 are instantaneous or take place in zero time. As a matter of fact an inspection of the transmitter and the receiver FSMs reveals that only at those states whose output or selection is **idle**, passage of time occurs. In other words, if q is the current state of any timer automata (TO, RO, T1, R1), the transition from q to $q+1$ is enabled only if the FSMs representing the transmitter, the receiver and the transmission channels, all have idle outputs or selections.

A timer state is reset to 0 if a transition whose enabling condition includes Reset–*timer*, is enabled. Note that by our notation, Reset–*timer* is not an enabling condition, that is to say, if the predicate only consists of Reset–*timer* then the corresponding transition is always enabled.

In addition to the mentioned FSMs and only for the sake of verification, we have to model the difference between transmitter and receiver bauds as yet another finite state process. This is due to the fact that the enabling condition of some of the transitions depends on the value of this difference; in particular, the receiver is not able to receive any messages if the difference of the bauds is not zero, even if the transmitter channel has delivered the message.

But our objectives are also defined in terms of the baud difference. Thus, we can use this FSM as a monitor to verify the protocol at the same time. Figure 5 shows this process, where an ERROR state, in which all the undesirable transitions terminate, has been added.

The transitions indicated here correspond to the relative base baud approach explained in section 2, but also apply to the single base baud case. Here,

$$\begin{aligned}
\mathbf{0} &\stackrel{\text{def}}{=} \text{(Trans. Baud-Rate} = \text{Rec. Baud-Rate)} \\
\mathbf{1} &\stackrel{\text{def}}{=} \text{(Trans. Baud-Rate} - \text{Rec. Baud-Rate} = \mathbf{1)} \\
-\mathbf{1} &\stackrel{\text{def}}{=} \neg(1) \\
\mathbf{B} &\stackrel{\text{def}}{=} \text{(Rec. Baud-Rate} = \text{Base Baud-Rate} \neq \text{Trans. Baud-Rate)} \\
-\mathbf{B} &\stackrel{\text{def}}{=} \neg(B)
\end{aligned}$$

To verify the protocol, we check that $\mathcal{L}(M)$, the language generated by the protocol (product of all the afore-mentioned FSMs) has no element which results in error, i.e.,

$$\nexists s \in \mathcal{L}(M) \quad \ni \quad \gamma(q_0, s) = q^* \quad \text{and} \quad q^* \Rightarrow (\text{BAUD\_DIFF: } \mathbf{error})$$

Or equivalently, all other states in the baud difference FSM form a cycle set for $\mathcal{L}(M)$. Yet in other words, if one considers the ERROR state as the final state, the protocol is verified if the language accepted by M is empty.

One may also check if for all selection sequences in $\mathcal{L}(M)$, the state corresponding to equal bauds, i.e.(BAUD_DIFF: 0), is infinitely often visited. However, the BAUDDIFF machine is not the appropriate monitor if one must verify that in each AT interval, the baud difference is zero during at least one control data transmission. For this task, a separate monitor process should be used. Figure 6 shows the finite state machine corresponding to this monitor.

We used COSPAN to verify the proposed protocol. All the mentioned finite state processes, except for the transmission channels, were programmed in the S/R language. Since the only role of the transmission channels is to add a delay to the system, we incorporated these delays, as well as the undeterministic loss of messages, in the transmitter and the receiver FSMs and eliminated the machines representing the transmission channels. In the program, we assumed

that the channel delay is negligible (zero) and that the receiver delay is 1 time unit (δt). We also considered the transmitter and the receiver processes to be asynchronous.

It should also be mentioned that in reality the timing constraints (time periods allocated to data transmissions etc.) depend on the absolute value of the baud, but in the program, we modeled these constraints to be dependent on the relative baud or the baud difference; if transmitter baud is greater then its waiting time is shorter relative to the receiver and so on. Since we assume that the AT interval is sufficiently long even for the worst case, this causes no loss of generality.

The two above-mentioned tasks were tested by using COSPAN, where both tasks were performed, thus, verifying our proposed protocol. For the COSPAN programs contact the authors.

4 Protocol Simulation

Simulation of the baud change protocol is done using CSIM simulation environment. CSIM is a process-oriented, discrete-event simulation package. We modeled the transmitter and the receiver as different processes and used standard mailboxes for message passings. We also modeled the channel as a separate process which uses random numbers to model the probability of errors. The channel process receives the messages and delivers them to the receiver mailbox, if it is not lost.

the parameters of the simulation were as follows:

- There are seven baud levels (19.2K-1M)
- Time-out is set to twice the sum of the transmission time and processing delay.
- As baud increases, number of possible data, request, and test packets increases.
- As baud increases, the probability of the packet loss increases.

In Figure 7, we show two trace of the simulation.

5 Conclusion

We presented in this paper a baud regulation protocol that can be used to increase or decrease the message transmission rate for the point-to-point communications. Since no baud regulation can be accomplished without some possibility of error (unless the transmission channels are ideal), the design objective was to regulate baud most efficiently and ensure at the same time that in each transmission interval, transmitter and receiver bauds are equal for some period of time during data transmissions. An equivalent discrete-time version of the protocol was verified by using the COSPAN software.

References

[1] P. Varaiya and S.E. Shladover. Sketch of an IVHS systems architecture. In *Proceedings of the Vehicle Navigation and Information Systems Conference,* pages 909–922, Dearborn, MI, October 20-23 1991.

[2] P. Varaiya. Smart cars on smart roads: Problems of control. *IEEE Transactions on Automatic Control,* 38(2), 1993.

[3] Farokh H. Eskafi. *Modeling and Simulation of Automated Highway System.* PhD thesis, Unversity of California, Berkeley, 1996.

[4] B.S.Y. Rao and P. Varaiya. Roadside intelligence for flow control in an IVHS. *Transportation Research Journal, part C,* 2(1):49–72, 1994.

[5] J. Lygeros and D. Godbole. Longitudinal control of the lead car of a platoon. *IEEE Transactions on Vehicular Technology,* 43(4):1125–1135, 1994.

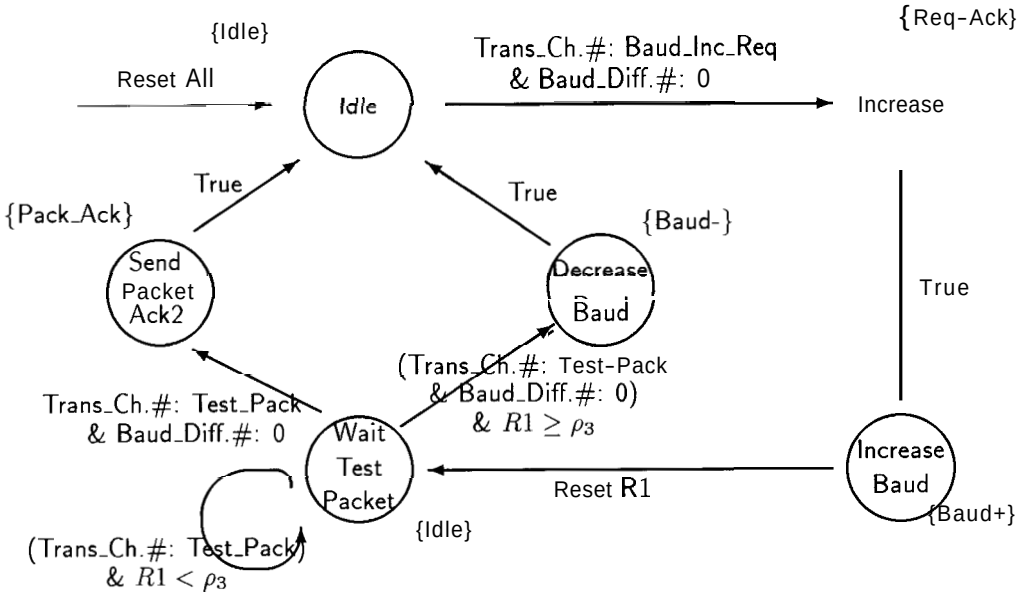
[6] H. Schwetman. *CSIM17 Reference Manual.* Mesquite Software, Inc., 3925 West Braker Lane, Austin, TX. 78759-5321, 1994.

[7] R. Alur and D. L. Dill. Automata for modeling real-time system. In *Proceedings of the 17th ICALP, Lecture Notes in Computer Science,* pages 322–335, 1990.

[8] R. Alur and R. P. Kurshan. Timing verification by successive approximation. AT&T Bell Labs Technical Memorandum, 1992.

RECEIVER

Increase Baud:



TRANSMITTER

Increase Baud:

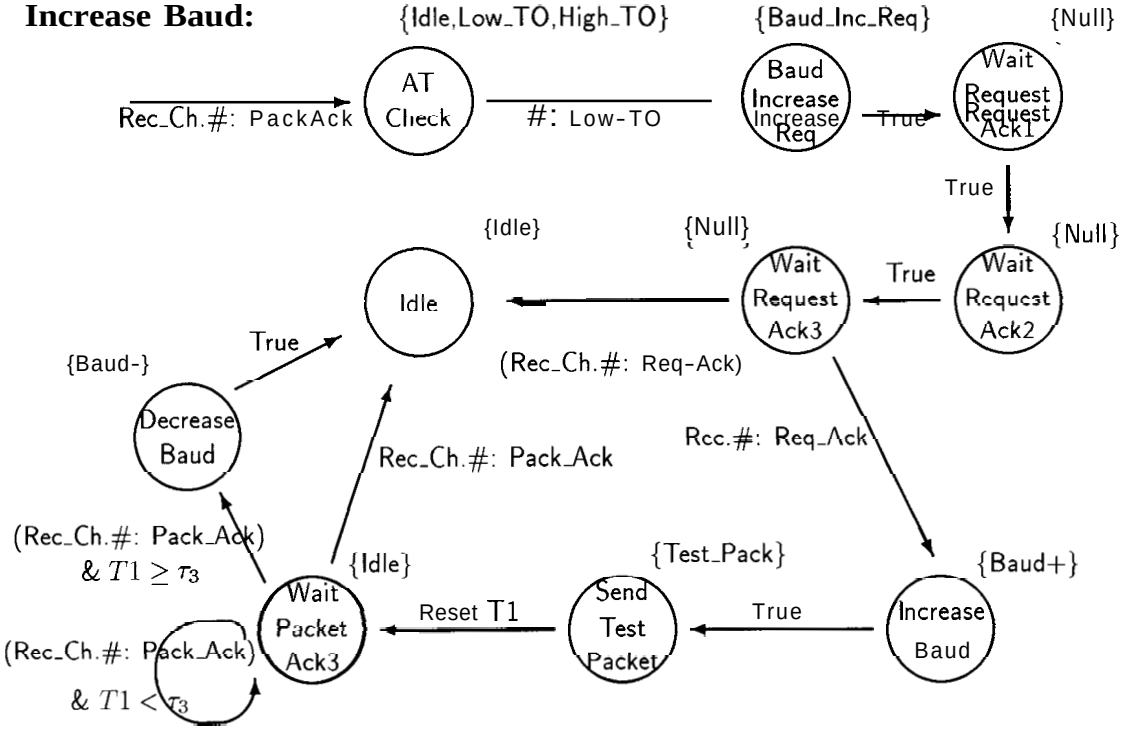
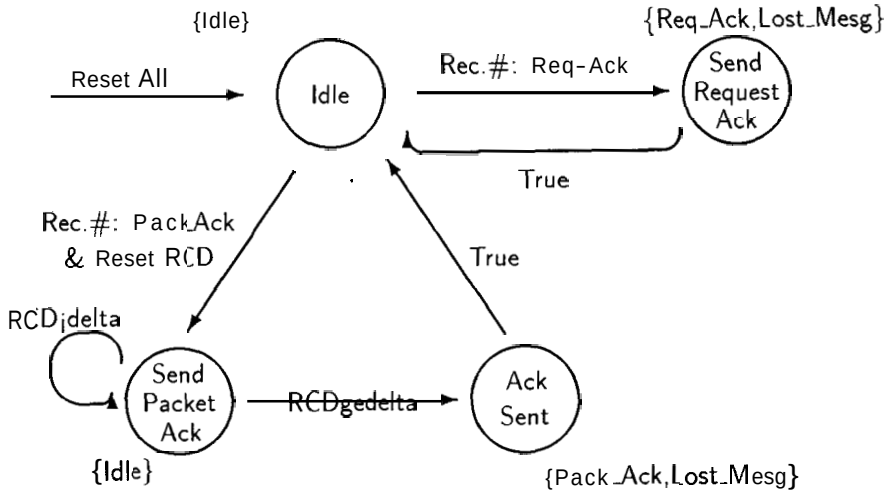


Figure 1: Transmitter and Receiver FSMs for Baud Increase

Rec. Channel:



Trans. Channel:

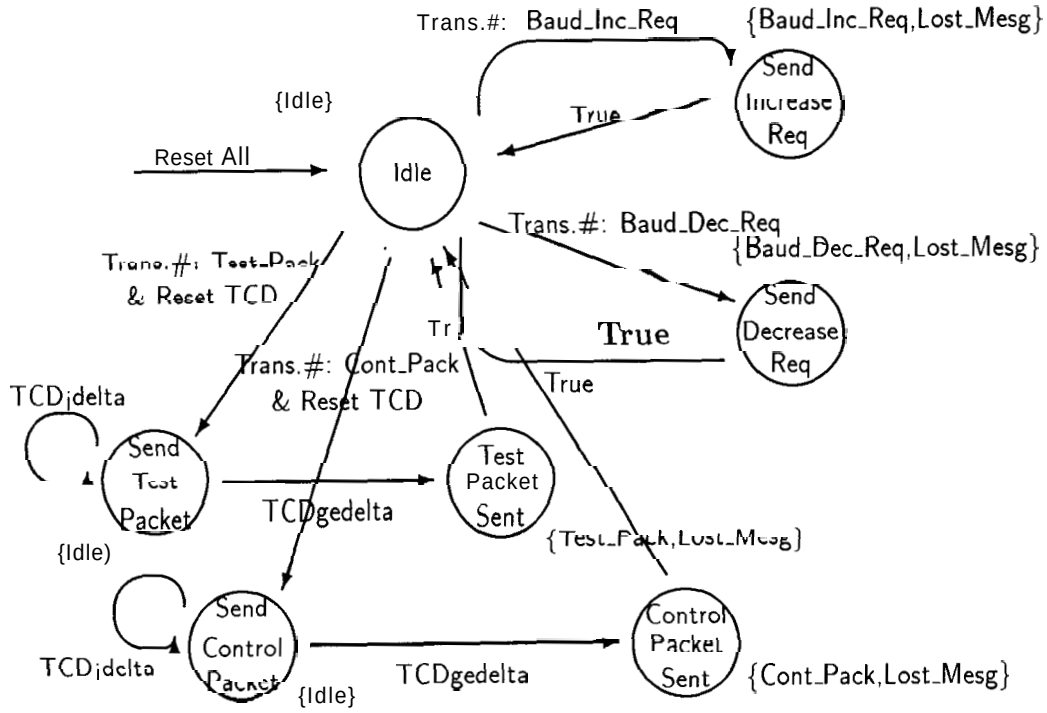


Figure 2: FSMs for Transmission Channels

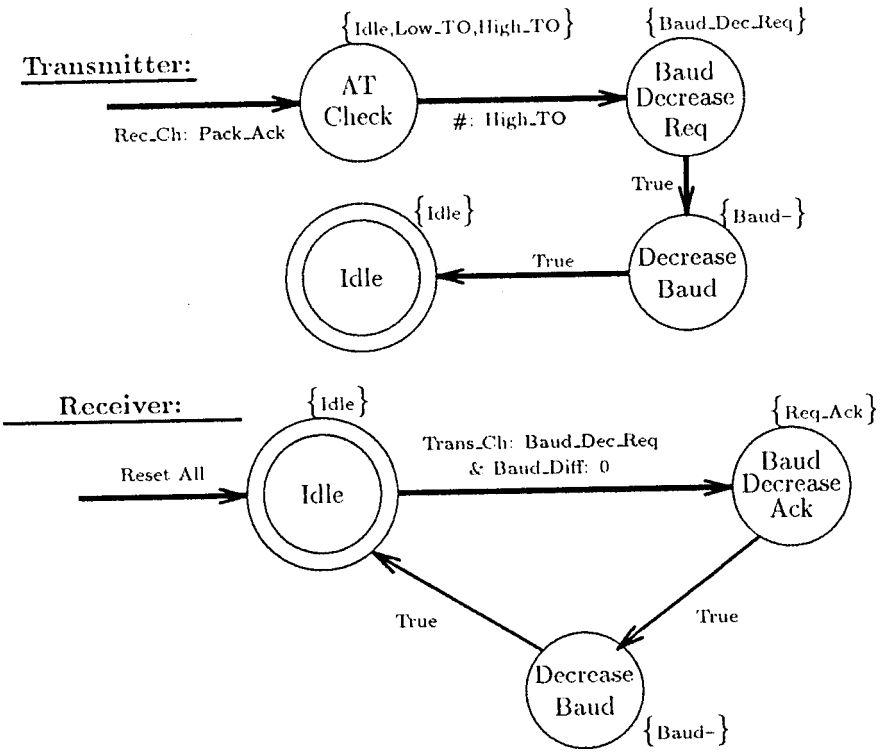


Figure *3:* Transmitter and Receiver FSMs for Baud Decrease

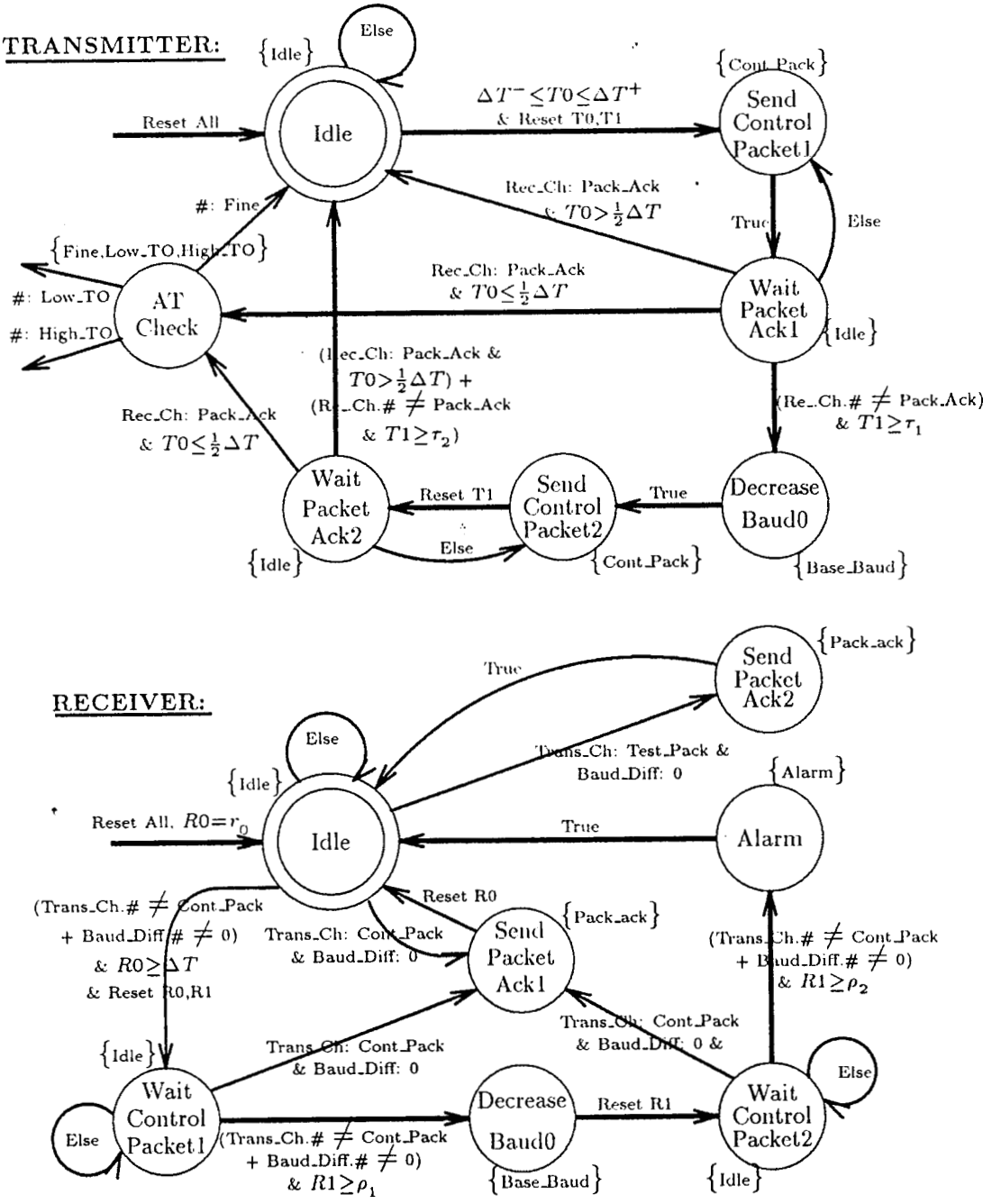
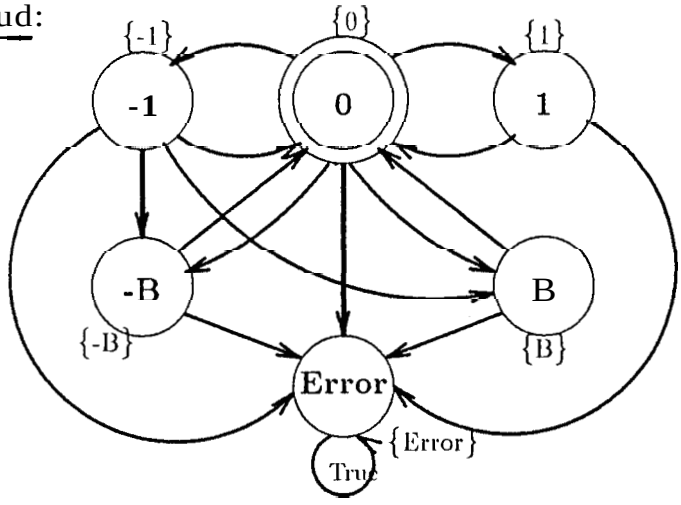


Figure 4: Transmitter and Receiver FSMs for Data Transmission/Baud Regulation

Trans. - Rec. Baud:



(0 → 1) = (Trans: Baud+ + Rec: Baud–) & (Trans.# ≠ Rec.#)
(0 → -1) = (Trans: Baud– + Rec: Baud+) & (Trans.# ≠ Rec.#) (1 → 0) = (0 → -1)
(-1 → 0) = (Rec: Baud– + Trans: Baud+) & (Trans.# ≠ Rec.#) + (Trans.#: Base-Baud) & (Rec.#: Base-Baud)
(0 → -B) = Trans: Base-Baud & (Trans.# # Rec.#) (B → 0) = (-1 → -U) = (0 → -B)
(0 → B) = Rec: Base-Baud & (Trans.# ≠ Rec.#) (MB → 0) = (-1 → B) = (0+ B)
(0 → Error) = Trans: Baud± & Rec: Base-Baud + Rec: Bautlf & Trans: Base-Baud
+ (Trans: Baud+ & Rec: Baud–) + (Trans: Baud- & Rec: Baud+)
(1 → Error) = (0 → Error) + Trans: Baud+ + Rec: Baud– + Rec: Alarm + Trans: Base-Baud + Rec: Base-Baud
(-1 → Error) = (0 → Error) + Trans: Baudf + Rec: Baud+ + Rec: Alarm
(MB → Error) = Trans: Bautlf + Rec: Baudf + Trans: Base-Baud & (Trans.# ≠ Rec.#)
(B → Error) = Trans: baud± + Rec: Baudf + Rec: Base-Baud + Rec: Alarm

Figure 5: The Baud Difference FSM

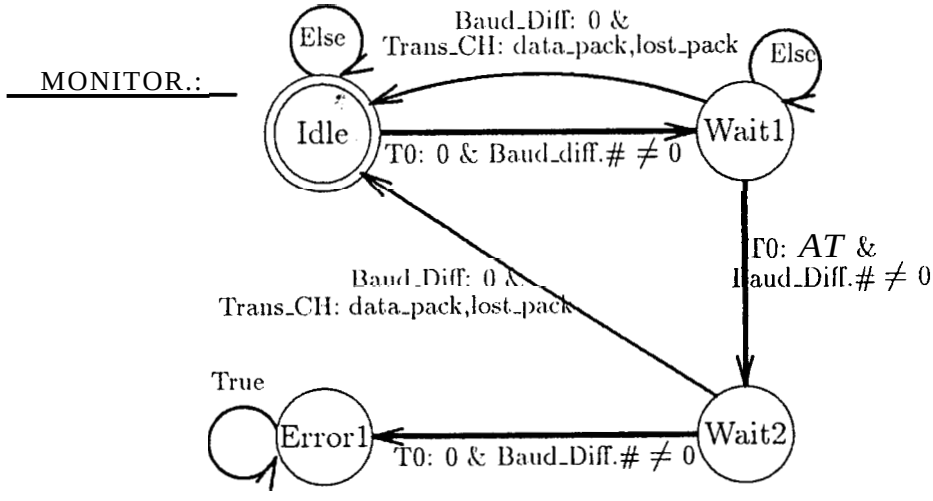


Figure 6: The Monitor Process

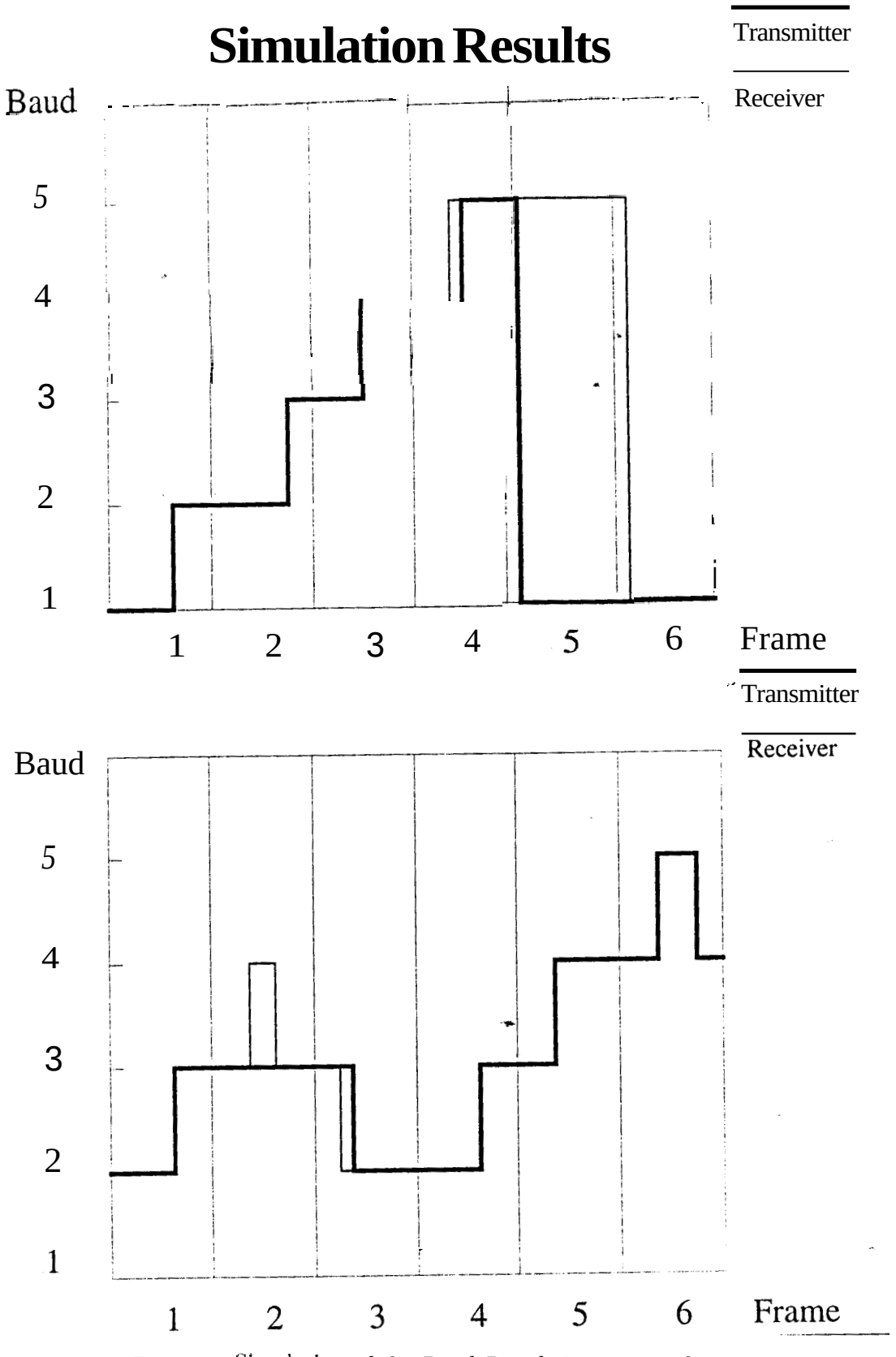


Figure 7: Simulation of the Baud Regulation Protocol