Chapter 10
Timed Coloured Petri Nets

This chapter shows how timing information can be added to CPN models. This makes it possible to evaluate how efficiently a system performs its operations and it also makes it possible to model and validate real-time systems [78], where the correctness of the system relies on the proper timing of the events. With a timed CPN model, performance measures such as maximum queue lengths and mean waiting times can be calculated. Also, we may, for example, verify whether the operation of a real-time system meets required deadlines.

It should be noted that it is often beneficial for the modeller to start by constructing and validating an untimed CPN model. In this way, the modeller can concentrate on the functional correctness of the system before worrying about timing issues. For the protocol described in Sect. 2.4, we saw that it was possible to describe the existence of time-related system features, such as retransmissions, without explicitly specifying concrete waiting times or the durations of the individual events. This is often the case, and it is a sound design strategy to try to make the functional correctness of a system independent of concrete assumptions about execution times and waiting times.

CPN models can be used to validate both the functional correctness and the performance of a system. This saves a lot of time, because we do not need to construct two totally independent models of the system. Instead, a single model or, more often, two closely related models are used. There exist a number of modelling languages that are in widespread use for performance analysis of systems, for example languages based on queueing theory [9]. However, most of these modelling languages turn out to be rather useless when it comes to modelling and validation of the functional properties of systems. Some of these languages are also unable to cope with performance analysis of systems which have an irregular behaviour. In this chapter, the concept of time in CP-nets is presented using a non-hierarchical CPN model as an example. The timing constructs also apply to hierarchical CP-nets, and CPN Tools supports the simulation and analysis of timed hierarchical CP-nets. The concept of time in CP-nets is one out of many time concepts that have been developed in the context of Petri Nets [90].

K. Jensen, L.M. Kristensen, *Coloured Petri Nets*, DOI 10.1007/b95112_10,

Section 10.1 presents a first timed CPN model of our protocol and introduces the basic constructs of timed CPN models. Section 10.2 considers a second timed CPN model of the protocol and introduces additional constructs of timed CPN models. Section 10.3 discusses basic state space analysis of timed CPN models and Sect. 10.4 presents a special case of the equivalence method presented in Sect. 8.4 that can be used to obtain a finite state space for any timed CPN model where the corresponding untimed CPN model has a finite state space.

10.1 First Timed Model of the Protocol

Consider Fig. 10.1, which contains a timed version of the CPN model of the protocol described in Sect. 2.4. It is easy to see that the CPN model is very closely related to the untimed CPN model in Fig. 2.10. The colour set definitions and variable declarations for the CPN model are given in Fig. 10.2.

The main difference between timed and untimed CPN models is that the tokens in a timed CPN model, in addition to the token colour, can carry a second value, called a *timestamp*. This means that the marking of a place where the tokens carry timestamps is now a *timed multiset*, specifying the elements in the multiset together with their timestamps. Furthermore, the CPN model has a *global clock*, representing *model time*. The distribution of tokens on the places, together with their timestamps and the value of the global clock, is called a *timed marking*. In a hierarchical timed CPN model there is a single global clock, shared among all of the modules.

The timestamps in CPN Tools are non-negative integers belonging to a CPN ML type called `TIME`. The timestamp specifies the time at which the token is *ready* to be

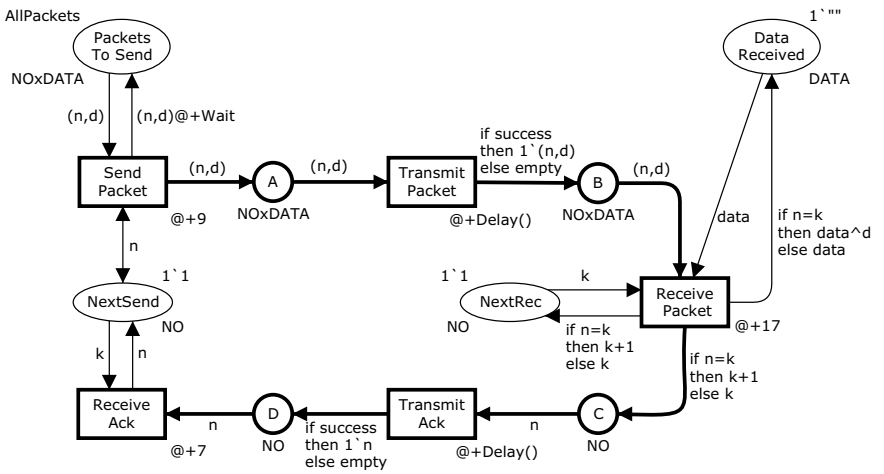

Fig. 10.1 Timed CPN model of the protocol

```
colset NO      = int timed;
colset DATA    = string timed;
colset NOxDATA = product NO * DATA timed;
colset BOOL    = bool;

var n, k    : NO;
var d, data : DATA;
var success : BOOL;
```

Fig. 10.2 Colour sets and variables for the timed CPN model shown in Fig. 10.1

used, i.e., the time at which it can be removed by an occurring transition. A colour set is declared to be timed using the CPN ML keyword `timed`. It can be seen from Fig. 10.2 that all places in Fig. 10.1 have *timed colour sets*.

The initial marking of the timed CPN model of the protocol is shown in Fig. 10.3. The colours of the tokens are the same as in the initial marking of the untimed CPN model of the protocol, but now the tokens also carry timestamps. As an example, the initial marking of the place PacketsToSend is

```
1`(1,"COL")@0 +++
1`(2,"OUR")@0 +++
1`(3,"ED ")@0 +++
1`(4,"PET")@0 +++
1`(5,"RI ")@0 +++
1`(6,"NET")@0
```

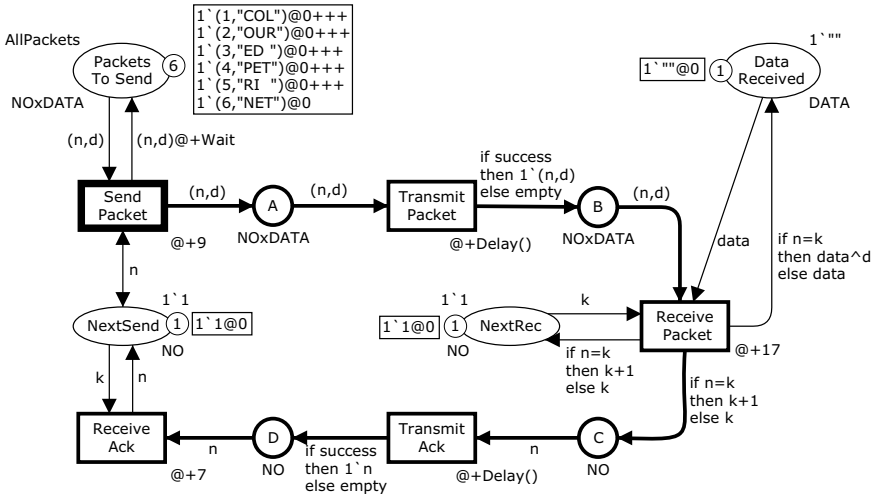

Fig. 10.3 Initial marking M_0 of the timed protocol model

The timestamps of tokens are written after the symbol `@`, which is pronounced 'at'. In this case, all tokens carry the timestamp 0. The operator `+++` takes two timed multisets as arguments and returns their union. All other tokens in the initial marking also carry the timestamp 0. The value of the global clock in the initial marking is also 0. The initial markings of all places are specified as an (untimed) multiset. CPN Tools will automatically attach a timestamp 0 if the initial marking inscription of a place with a timed colour set does not explicitly specify the timestamps of the tokens.

In the initial marking M_0, there is only one binding element that has the required tokens on its input places. This is the transition SendPacket, with the binding ⟨`n=1`, `d="COL"`⟩. To occur, this binding element needs the presence of a token with colour `1` on the place NextSend and the presence of a token with colour `(1,"COL")` on the place PacketsToSend. This is determined by the input arc expressions by means of the enabling rule explained in Chap. 2. We see that the two tokens that are needed by NextSend exist on the input places and that both of them carry the timestamp `0`, which means that they can be used at time `0`. Hence, the transition can occur at time `0`. When the transition occurs, it removes the two tokens from the input places and adds a token to each of the three output places. The colours of these tokens are determined from the output arc expressions by means of the occurrence rule explained in Chap. 2. However, it is also necessary to calculate the timestamps to be given to the three output tokens. This is done by using *time delay inscriptions* attached to the transition and/or to the individual output arcs. A time delay inscribed on a transition applies to all output tokens created by that transition, whereas a time delay inscribed on an output arc applies only to tokens created at that arc. In Fig. 10.3 we have associated a constant time delay expression `@+9` with the transition SendPacket. The outgoing arc to PacketsToSend has a constant time delay expression `@+Wait`, where `Wait` is a symbolic constant defined as

```
val Wait = 100;
```

The arc expressions on the output arcs to the places A and NextSend have no separate time delays. The timestamp given to the tokens created on an output arc is the sum of the value of the global clock, the result of evaluating the time delay inscription of the transition, and the result of evaluating the time delay inscription of the arc. Hence, we conclude that the tokens added to the places NextSend and A will receive the timestamp

$$0+9+0=9$$

The first 0 is the time at which the transition occurs as given by the global clock, the 9 is the time delay inscribed on the transition, and the second 0 is the time delay on the output arc (since there is no time delay on the output arc). Intuitively, this means that the execution of the 'send packet' operation has a duration of 9 time units.

The arc expression on the output arc to the place PacketsToSend has a separate time delay: `@+Wait`. This means that the token added to PacketsToSend will receive the timestamp

$$0 + 9 + 100 = 109$$

The 0 is the time at which the transition occurs, the 9 is the time delay inscribed on the transition, and the 100 is the time delay inscribed on the output arc. Intuitively, this represents the fact that we do not want to resend data packet number 1 until time 109, i.e., until 100 time units after the end of the previous send operation. This is achieved by giving the token for data packet number 1 the timestamp 109, thus making it unavailable until that moment of time. However, it should be noticed that data packet number 2 still has a timestamp 0. Hence, it will be possible to transmit this data packet immediately, if an acknowledgement arrives before time 109. When SendPacket occurs at time 0, we reach the marking M_1 shown in Fig. 10.4.

In the marking M_1, there are three binding elements that have the needed tokens on their input places:

SP1 = (SendPacket, ⟨n=1, d="COL"⟩)
TP1$^+$ = (TransmitPacket, ⟨n=1, d="COL", success=true⟩)
TP1$^-$ = (TransmitPacket, ⟨n=1, d="COL", success=false⟩)

SP1 can occur at time 109 since it needs a token with timestamp 109 and a token with timestamp 9. However, TP1$^+$ and TP1$^-$ can already occur at time 9, because they need a token with timestamp 9. Since TP1$^+$ and TP1$^-$ are the first binding elements that are ready to occur, one of these will be chosen. This means that SP1 cannot occur in the marking M_1, and hence SendPacket has no thick border in Fig. 10.4. The chosen binding element will occur as soon as possible, i.e., at time 9. Only one of them will occur, since the two binding elements are in conflict with each other.

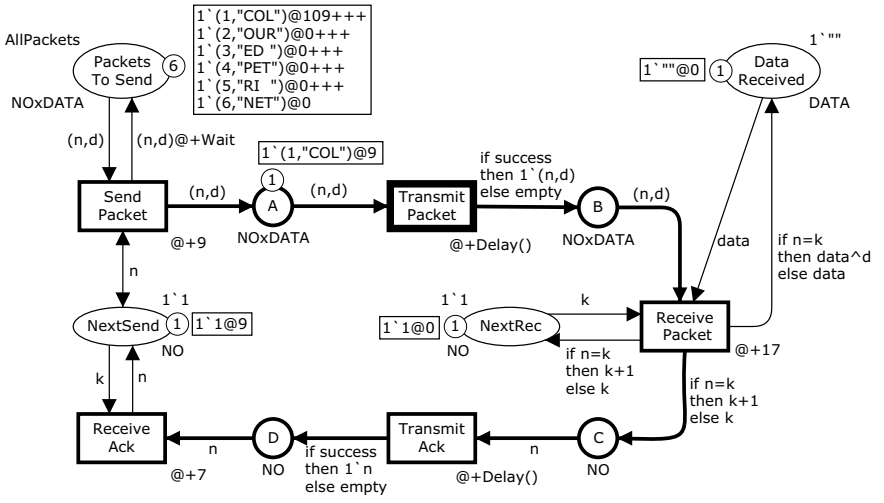

Fig. 10.4 Marking M_1 reached when SendPacket occurs at time 0 in M_0

Assume that TP1$^+$ is chosen to occur. It will remove the token from place A and add a token to place B. The timestamp of this token will be the sum of the time at which the transition occurs (9) and the evaluation of the time delay expression `@+Delay()` inscribed on the transition. The function `Delay` takes a unit, written `()`, as an argument and is defined as follows:

```
fun Delay () = discrete(25,75);
```

The function `discrete` is a predefined function that provides a discrete uniform distribution over the closed interval specified by its arguments. This means that `Delay()` returns an integer from the interval [25, 75] and that all numbers in the interval have the same probability of being chosen. Intuitively, this represents the fact that the time needed to transmit a packet over the network may vary between 25 and 75 time units owing to the load on the network, for example. Assume that `Delay()` evaluates to 38. We then reach the marking M_2 shown in Fig. 10.5.

In the marking M_2, there are two binding elements that have the needed tokens on their input places:

SP1 = (SendPacket, ⟨`n=1`, `d="COL"`⟩)
RP1 = (ReceivePacket, ⟨`n=1`, `d="COL"`, `k=1`, `data=""`⟩)

As before, SP1 can occur at time 109. However, RP1 can already occur at time 47, since it needs a token with timestamp 47 and two tokens with timestamp 0. Hence RP1 will be chosen and we reach the marking M_3 shown in Fig. 10.6.

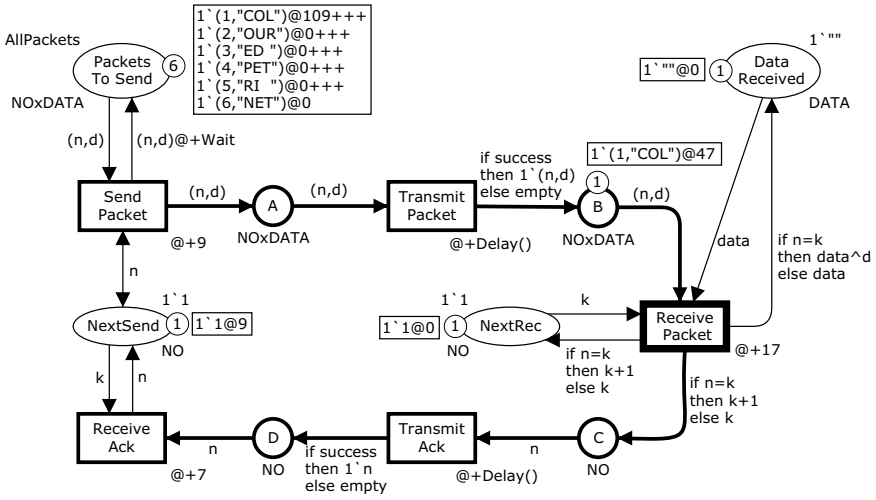

Fig. 10.5 Marking M_2 reached when TransmitPacket occurs at time 9 in M_1

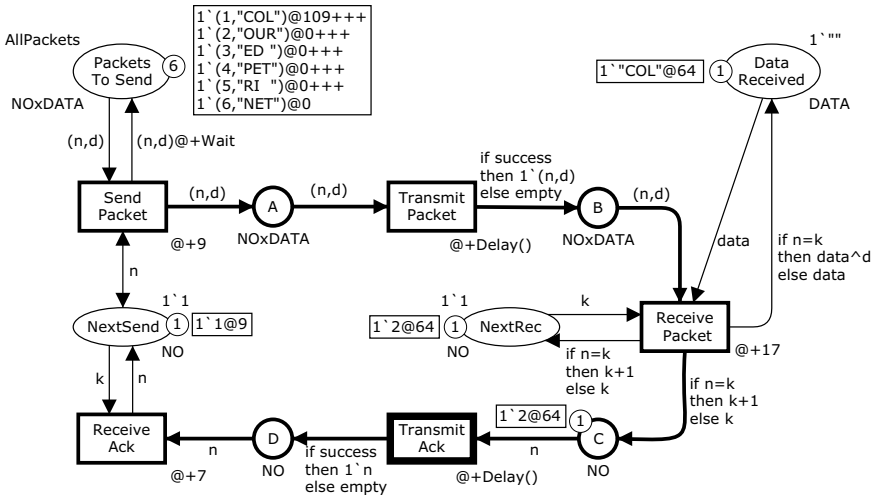

Fig. 10.6 Marking M_3 reached when ReceivePacket occurs at time 47 in M_2

In the marking M_3, there are three binding elements that have the needed tokens on their input places:

SP1 = (SendPacket, ⟨n=1, d="COL"⟩)
$TA2^+$ = (TransmitAck, ⟨n=2, success=true⟩)
$TA2^-$ = (TransmitAck, ⟨n=2, success=false⟩)

SP1 can occur at time 109. However, $TA2^+$ and $TA2^-$ can already occur at time 64 since they need a token with timestamp 64. Hence $TA2^+$ or $TA2^-$ will be chosen. Assuming that $TA2^+$ is chosen and that `Delay()` evaluates to 33 this time, we reach the marking M_4 shown in Fig. 10.7.

In the marking M_4, there are two binding elements that have the needed tokens on their input places:

SP1 = (SendPacket, ⟨n=1, d="COL"⟩)
RA2 = (ReceiveAck, ⟨n=2, k=1⟩)

SP1 can occur at time 109. However, RA2 can already occur at time 97, since it needs a token with timestamp 97 and a token with timestamp 9. Hence RA2 will be chosen, and we reach the marking M_5 shown in Fig. 10.8.

In the marking M_5, there is only one binding element that has the needed tokens on its input places:

SP2 = (SendPacket, ⟨n=2,d="OUR"⟩)

Hence SP2 will be chosen, and it will occur at time 104 because it needs a token with timestamp 104 from NextSend and a token with timestamp 0 from PacketsToSend. We then reach the marking M_6 shown in Fig. 10.9.

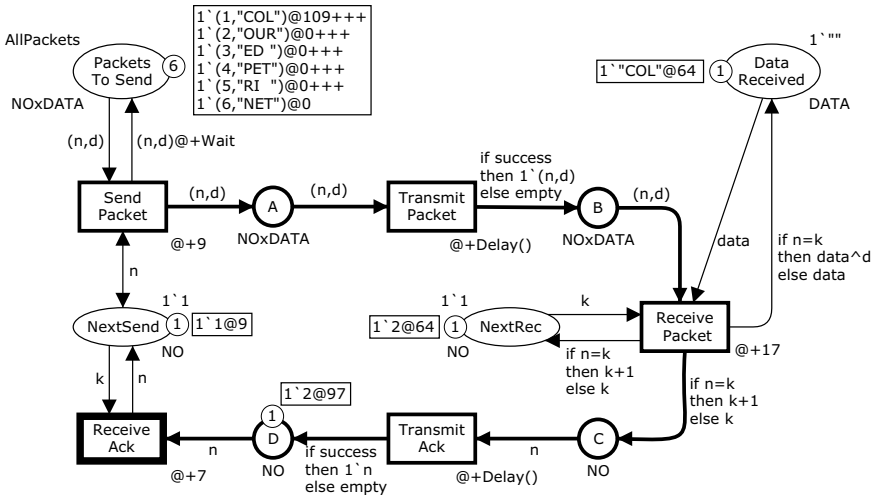

Fig. 10.7 Marking M_4 reached when TransmitAck occurs at time 64 in M_3

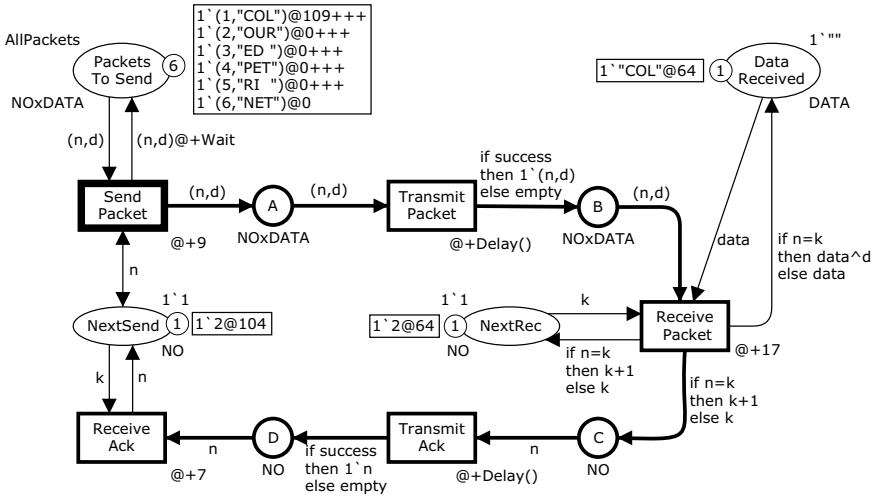

Fig. 10.8 Marking M_5 reached when ReceiveAck occurs at time 97 in M_4

In the occurrence sequence considered above, it turned out that no retransmission of data packet number 1 became possible. However, the two evaluations of `Delay()` in the time delay inscriptions of TransmitPacket and TransmitAck could have produced two larger values (e.g., 74 and 50 instead of 38 and 33). If this had been the case, TransmitAck would have produced a token on place D with timestamp 150 instead of 97, and we would have reached the marking M_4^* shown in Fig. 10.10 instead of the marking M_4 shown in Fig. 10.7.

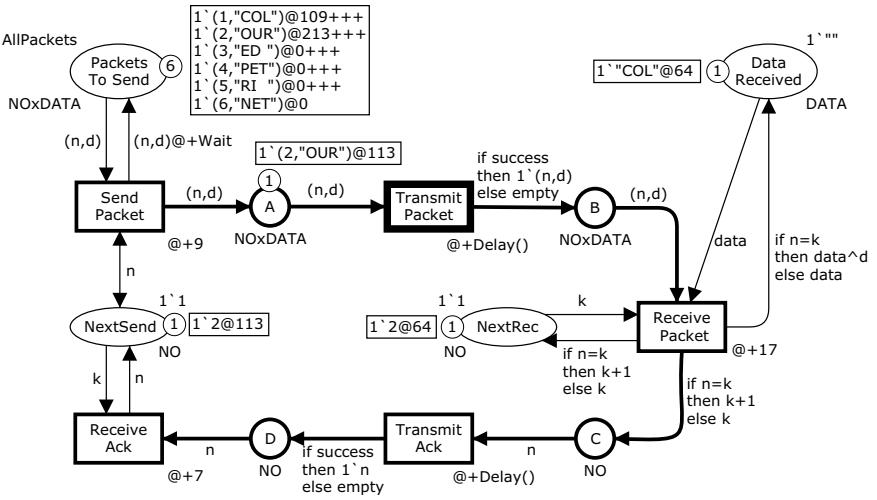

Fig. 10.9 Marking M_6 reached when SendPacket occurs at time 104 in M_5

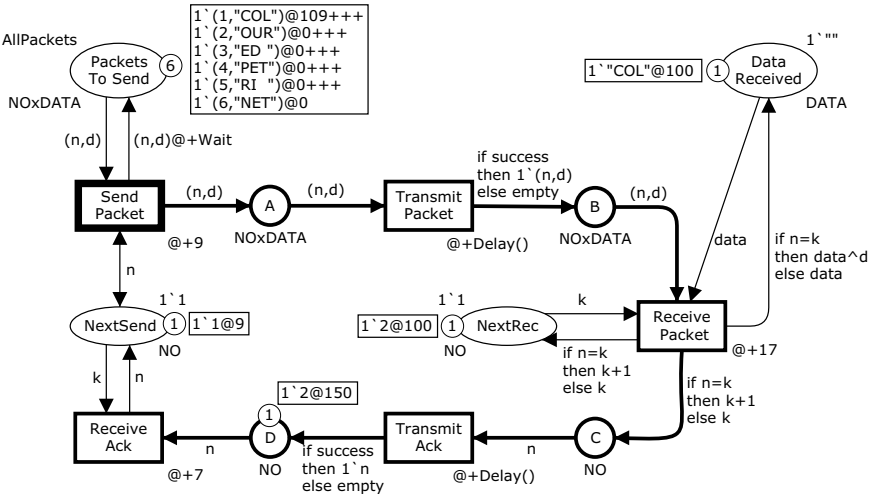

Fig. 10.10 Marking M_4^* reached when the network delays are larger

In the marking M_4^*, RA2 is ready to occur at time 150, i.e., later than SP1, which is ready to occur at time 109. Hence SP1 will be chosen instead of RA2, and we shall get a retransmission of data packet number 1.

Figure 10.11 shows a dead marking reached at the end of a simulation of the timed CPN model of the protocol. We can see the times at which the individual data packets would have been ready for the next retransmission (218, 2095, 2664, 2906, 3257, and 3499). Moreover, we can see that the last data packet was received at time 3357, and the last acknowledgement was received at time 3414. The CPN model is

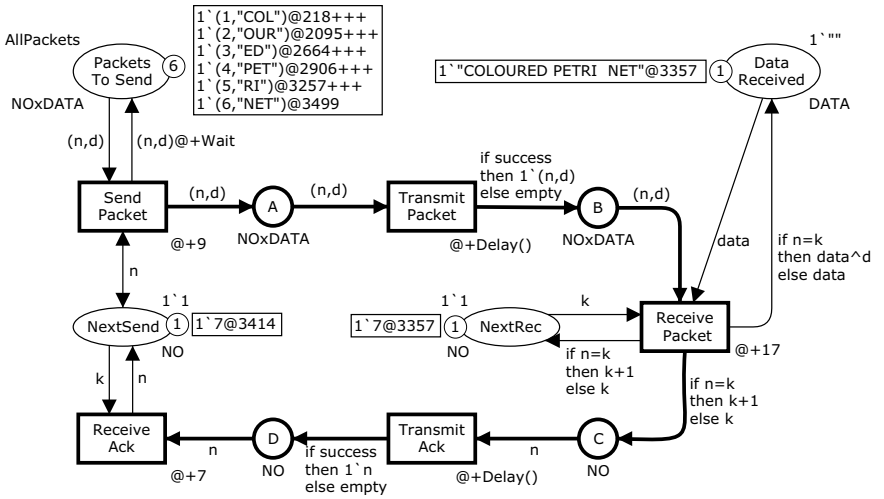

Fig. 10.11 Dead marking reached at the end of a simulation

non-deterministic and hence a second simulation would give other timestamps in the final marking, but the token colours would be the same.

In the timed markings shown above, there was never more than one token with a given colour on a place. As with an untimed CPN model, we can have several tokens with the same colour on a given place, and these may or may not have the same timestamps. As an example, consider the (non-reachable) marking of place D where we have four tokens with colour 2 and one of these has timestamp 405, two have timestamp 409, and one has timestamp 411, and we also have four tokens with colour 3, all having timestamp 410. This timed multiset is written as

```
1‘2@405 +++
2‘2@409 +++
1‘2@411 +++
4‘3@410
```

When we have several tokens with the same colour and these have different timestamps, we may have a situation where several tokens with a given colour are ready to be consumed. As an example, assume that we have an enabled binding element at time 409 that will remove a token with colour 2 from a place that has the timed multiset above as its marking. In this case, the three tokens with colour 2 and timestamps 405, 409, and 409 are ready to be consumed. In such situations, we remove the token with the largest possible timestamp. In this case, this means that one of the tokens with colour 2 and timestamp 409 will be removed. Removing the tokens with the largest possible timestamps ensures that a marking that can be reached by the occurrence of a step consisting of multiple binding element can also be reached by letting the binding elements occur sequentially in some arbitrary order, i.e., that Theorem 4.7 is also valid for timed CPN models. This means that

it is sufficient for the simulator in CPN Tools to consider only steps consisting of a single binding element.

In the timed CPN model considered above, all tokens carry a timestamp, since all colour sets of the places were declared to be timed. However, this is not in general the case. We allow the modeller to specify whether each individual colour set is timed or not. The tokens of timed colour sets carry timestamps, whereas the tokens of untimed colour sets do not. Tokens without timestamps are always ready to participate in occurrences of binding elements. As an example, assume that the timed CPN model of the protocol is modified such that the tokens on NextSend carry no timestamps while all other tokens do carry timestamps. To have timestamps on C, D, and NextRec and no timestamps on NextSend, we use the untimed colour set `INT` for NextSend, while we use the timed colour set `NO` for the places C, D, and NextRec. The initial marking of the modified CPN model is shown in Fig. 10.12. This model behaves in a way similar to the timed CPN model shown in Fig. 10.1. However, it is now possible for SendPacket and ReceiveAck to occur at the same model time, i.e., immediately after each other, since access to NextSend now takes zero time. This represents a situation in which the sender can perform several SendPacket and ReceiveAck operations at the same time, where we consider the occurrence of the corresponding transition to model the beginning of the operation. In the original model in Fig. 10.1, the SendPacket and ReceiveAck operations had to wait for the timestamp on the place NextSend and hence they could not occur at the same moment of model time.

The execution of a timed CPN model is controlled by the global clock, and works in a way similar to the event queue found in many simulation engines for discrete event simulation. The model remains at a given model time as long as there are binding elements that are *colour enabled* and *ready*. A binding element is colour

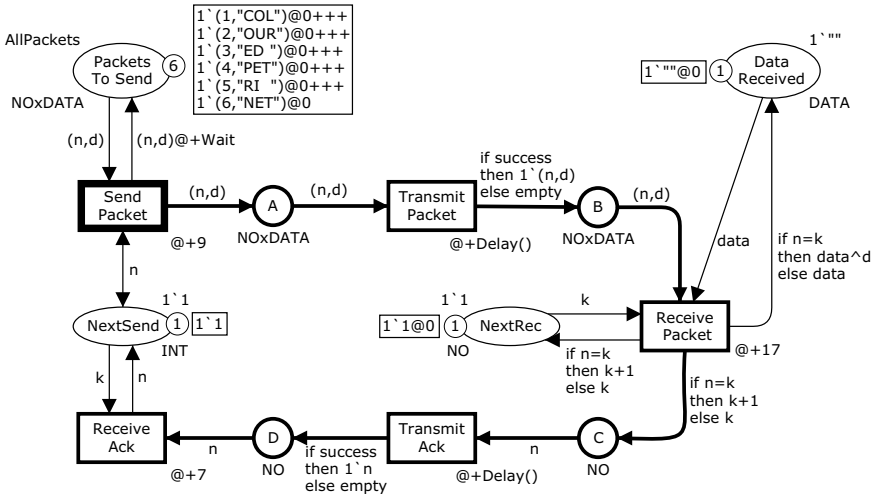

Fig. 10.12 CPN model with untimed colour set for NextSend

enabled if the required tokens are present on the input places. A binding element is *ready* for execution if these tokens have timestamps which are old enough, i.e., less than or equal to the current value of the global clock. Hence, in a timed CPN model an enabled binding element must be both colour enabled and ready in order to be able to occur. When there are no longer such binding elements to be executed, the simulator advances the clock to the next earliest model time at which binding elements can be executed. Each marking exists in a closed interval of model time, which may be a point, i.e., a single moment of time. As with untimed CPN models, we may have conflicts and concurrency between binding elements, and binding elements may be concurrent with themselves, but only if the binding elements are ready to be executed at the same moment of time.

The standard behavioural properties of timed CPN models are defined in a way similar to that for the untimed case. For multiset bounds and home markings/spaces, we consider the untimed markings of the places, i.e., we ignore the timestamps of tokens. The algorithms used by the CPN state space tool for computing the contents of the state space report are similar to those used for untimed CPN models.

A timed CPN model can always be transformed into an untimed CPN model by making all colour sets untimed, removing all timestamps from initialisation functions, and removing all time delay inscriptions on arcs and transitions. The possible occurrence sequences of the timed CPN model always form a subset of the occurrence sequences of the corresponding untimed CPN model. This means that the time delay inscriptions merely enforce a set of additional constraints on the execution of the CPN model that cause the binding elements to be chosen in the order in which they become ready for execution. Turning an untimed CPN model into a timed model cannot create new behaviour in the form of new occurrence sequences. This supports the soundness of our advice at the beginning of this chapter: start by investigating the functionality by means of an untimed CPN model. The timing related to events can then be considered afterwards.

The occurrence of a transition is instantaneous, i.e., takes no time. However, as shown in the protocol example above, it is easy to model a situation where some actions in a system have a non-zero duration. This is done by giving the output tokens created by the corresponding transition timestamps that prevent the tokens from being used until the time at which the action has finished. As an example, TransmitPacket cannot occur until 9 time units after the occurrence of SendPacket. This represents the fact that the action of sending a data packet takes 9 time units.

Instead, we could have chosen to allow the occurrence of a transition to have a non-zero duration. We could then remove the input tokens at the moment when the occurrence begins and add the output tokens when the occurrence ends. However, such an approach would make the relationship between a timed CPN model and its corresponding untimed CPN model much more complex. Now there would be many reachable markings in the timed CPN model which would be unreachable in the untimed CPN model because they corresponded to situations where one or more transitions were halfway through their occurrence, having removed tokens from the input places but not yet having added tokens to the output places.

The time values (i.e., timestamps and model time) considered above all belong to the set of integers. It is straightforward to generalise the concept of time in CP-nets such that time values belonging to the set of reals can be used, but the current version of CPN Tools supports only integer time values.

10.2 Second Timed Model of the Protocol

It turns out that there are situations where it is useful to allow a transition to remove a token from one of its input places ahead of time, i.e., at a moment of model time that lies before the timestamp carried by the token. As an example, consider Fig. 10.13, where we have performed a more detailed modelling of the operations in the sender, in particular the mechanism for timing the retransmission of data packets. The transition SelectNext is enabled in the initial marking M_0, and its occurrence models the situation where the sender selects the next data packet for transmission. There is no time delay inscription associated with this transition, since we consider the duration of selecting the next data packet to be insignificant. A similar remark applies to the transitions TimeOut and StopTimer. In this variant of the sender, we have only associated time delay inscriptions with the transitions SendPacket and ReceiveAck.

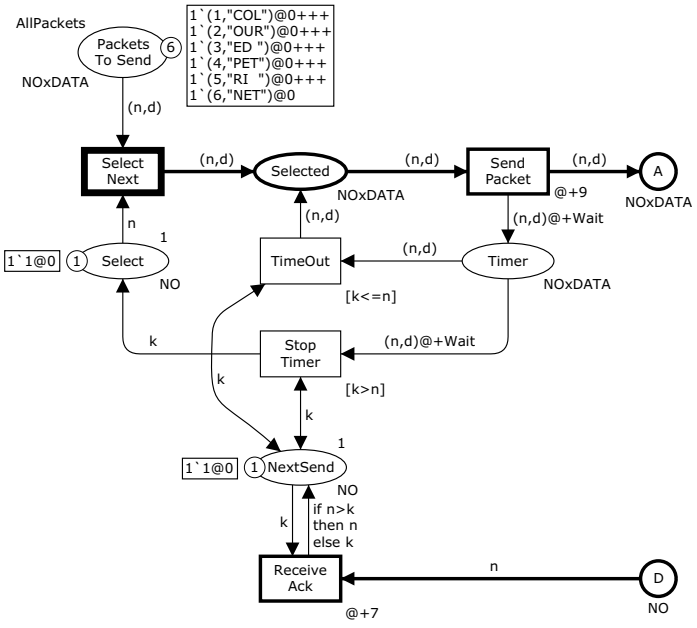

Fig. 10.13 Variant of the sender, in the initial marking M_0

When the transition SelectNext occurs in the initial marking shown in Fig. 10.13, we obtain the marking M_1 shown in Fig. 10.14, in which the first data packet has been removed from the place PacketsToSend and put on the place Selected.

The global clock will not be increased, since the transition SendPacket is enabled at time 0. When SendPacket occurs, we reach the marking M_2 shown in Fig. 10.15. The time delay inscription `Wait` on the arc from SendPacket to Timer is used to set the expiration time for a timer modelled by the place Timer. This ensures that the transition TimeOut cannot occur until `Wait` time units after the previous send operation. The guard of the transition TimeOut ensures that it can occur only if the current data packet has not been acknowledged, since the place NextSend always contains the highest sequence number received in an acknowledgement. If no acknowledgement for the data packet currently being sent arrives before time 109, the transition TimeOut will occur, leading back to a marking of the sender similar to the one shown in Fig. 10.14 in which the data packet can be sent once more (but with a higher timestamp).

Assume now that an acknowledgement 2 arrives at place D at time 94, as shown in Fig. 10.16. The transition ReceiveAck will occur at time 94, leading to the marking M_4 shown in Fig. 10.17.

In this marking, the transition StopTimer will be enabled at time 101 despite the timestamp 109 on the place Timer. This is achieved by using a time delay inscription on the arc from Timer to StopTimer, i.e., on an input arc of a transition. Until now we have used time delay inscriptions only on transitions and output arcs to specify what

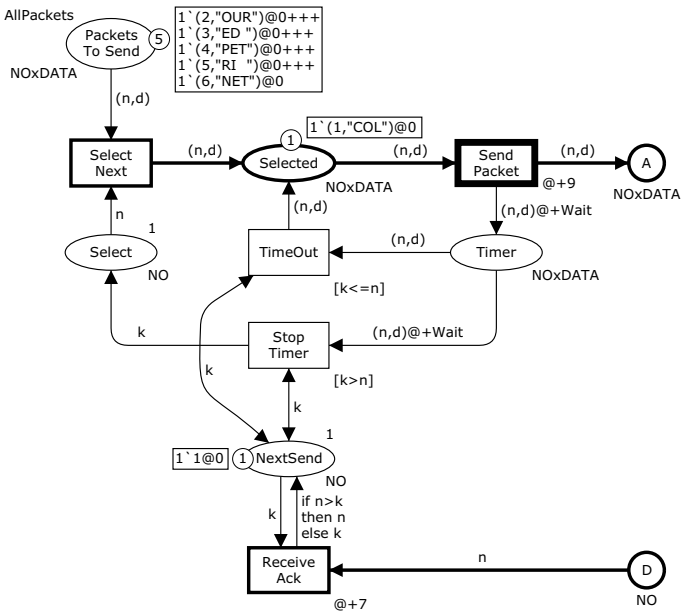

Fig. 10.14 Marking M_1 reached when SelectNext occurs at time 0 in M_0

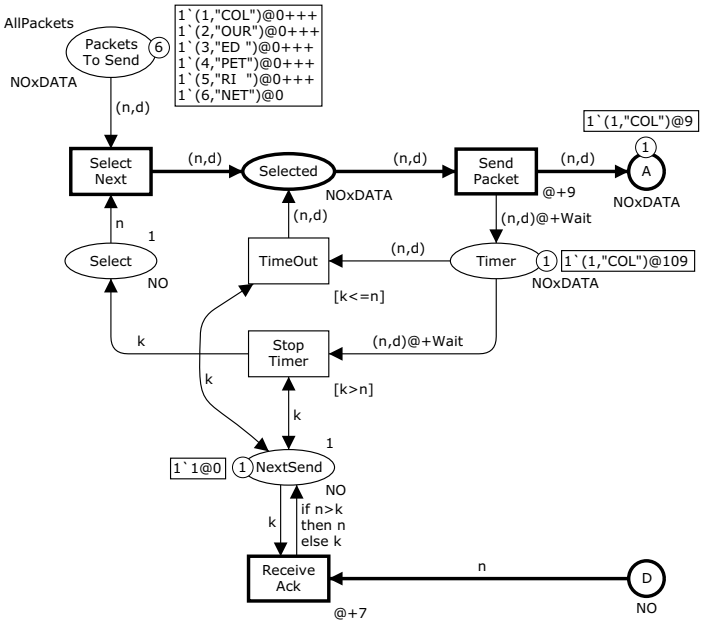

Fig. 10.15 Marking M_2 reached when SendPacket occurs at time 0 in M_1

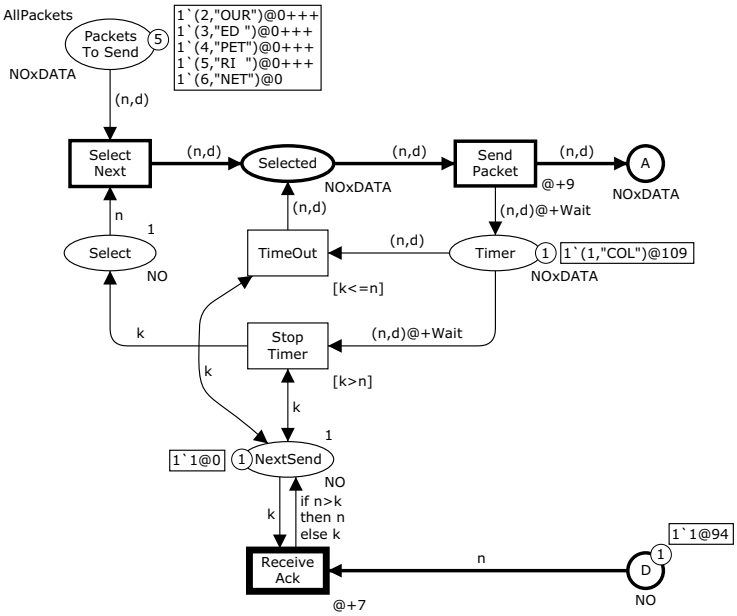

Fig. 10.16 Marking M_3, where an acknowledgement arrives at time 94

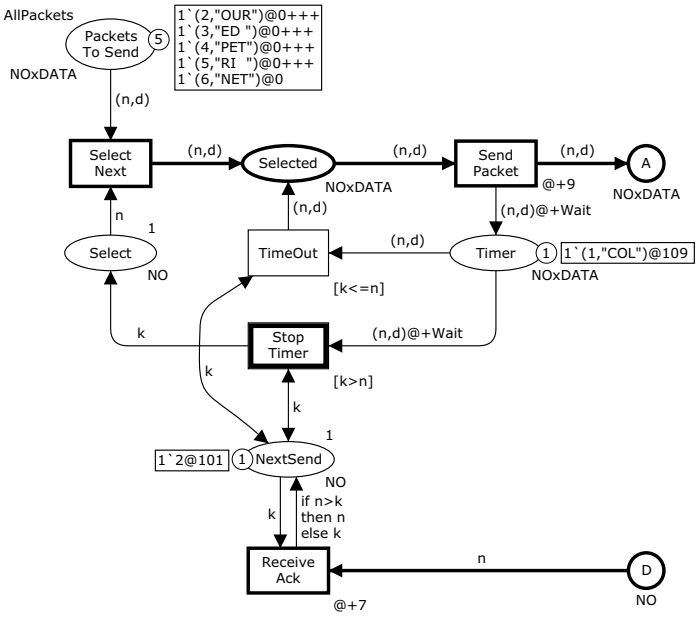

Fig. 10.17 Marking M_4 reached when ReceiveAck occurs at time 94 in M_3

should be added to the current value of the global clock to obtain the timestamp of the token produced on an output place. When a time delay is used on an input arc, the time delay inscription specifies how far ahead of time the transition can remove tokens from an input place. In this case we want to remove the token on the place Timer as soon as we have received an acknowledgement for the data packet currently being sent, and in this way ensure that the transition TimeOut will not occur, causing the retransmission of the data packet. In this way, we can stop the timer represented by the token on the place Timer. We have used `Wait` in the time delay inscription on the input arc from Waiting to TimeOut since the timer will have to be disabled at most `Wait` time units ahead of time. When the transition StopTimer occurs in M_4, we reach the marking M_5 shown in Fig. 10.18, in which a token has been put on the place Select, indicating that the current data packet has been acknowledged and that the sender is ready to select and send the next data packet.

When a time delay inscription is used on a double-headed arc in a timed CPN model, it is a shorthand for an arc in both directions with the same arc expression, *including* the time delay inscription. This means that time delay inscriptions on double-headed arcs must be used with care to avoid unintentionally removing tokens ahead of time.

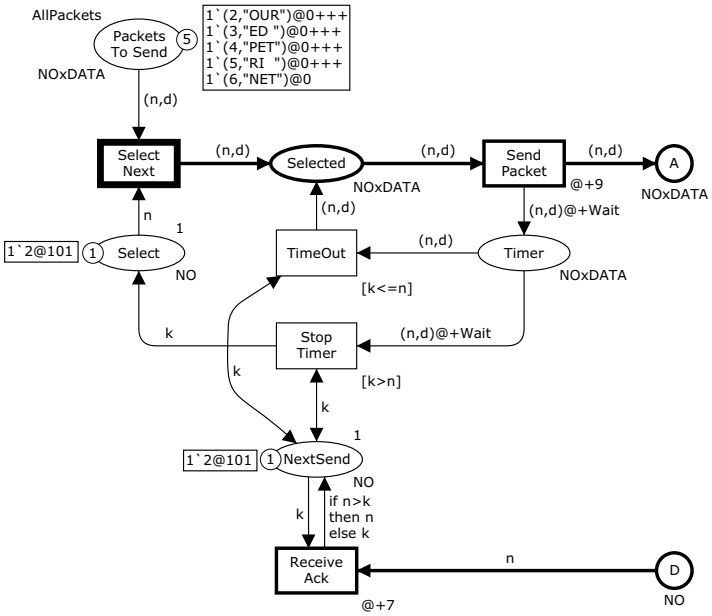

Fig. 10.18 Marking M_5 reached when StopTimer occurs at time 101 in M_4

10.3 State Space Analysis of Timed Models

The state space of a timed CPN model is defined in a way similar to that for untimed CPN models, except that each state space node now represents a timed marking, i.e., the value of the global clock and timed multisets specifying the markings of the places.

In the section above, we have seen that each occurrence sequence in a timed CPN model corresponds to an occurrence sequence in the corresponding untimed CPN model, but usually not the other way around, since the timestamps of the tokens put additional constraints on enabling. This reduces the outdegree (i.e., the number of outgoing arcs) of the nodes in the timed state space. Nevertheless, the timed state space may be larger than the untimed state space. The reason for this is that the nodes in the timed state space represent timed markings and include the global clock and timestamps. Hence, two timed markings can be different even if the corresponding untimed markings are identical (see M_4 and M_4^* in Figs 10.7 and 10.10). This means that the timing information makes more markings distinguishable and hence contributes to the presence of more nodes in the state space. The structure of the state space for a timed CPN model is therefore, in general, different from the structure of the state space for the corresponding untimed CPN model. As we

shall see below, this also means that the timed state space can be infinite, even if the state space of the corresponding untimed CPN model is finite. Furthermore, the difference in structure also means that the timed CPN model and the corresponding untimed CPN model will satisfy different behavioural properties.

The timed CPN model presented in the previous section is not directly suited for full state space analysis for two reasons. The first reason is that we want to limit the number of packets that can be present simultaneously on the network. The second reason is that we have used the function `Delay` to model the delay associated with transmitting packets on the network. This means that the state space is not well defined, since the set of reachable markings depends on the values returned by `Delay`, which in turn depends on a random number generator. Two consecutive state space generations may therefore result in different state spaces. This problem applies to the use of functions that return random values independently of whether the model is timed or not.

Figure 10.19 shows the initial marking of a variant of the timed protocol where we have resolved the two issues above. The place Limit is used to limit the number of packets simultaneously present on the network. Furthermore, we have introduced the two places DelayTP and DelayTA, connected to the transitions TransmitPacket and TransmitAck, respectively. The variable `delay` is of type `INT`, and the constant `Delays` is defined as

```
val Delays = 1`25 ++ 1`50 ++ 1`75;
```

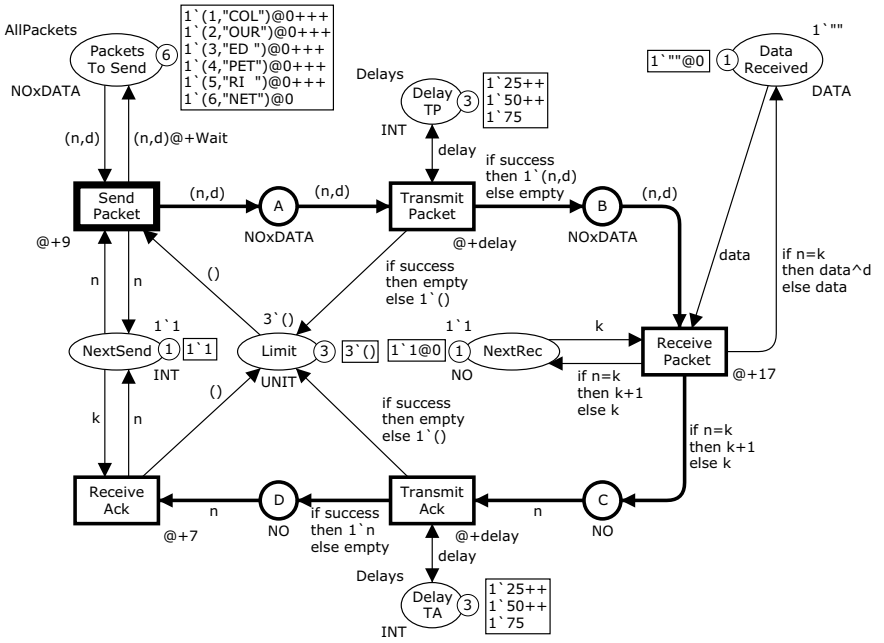

Fig. 10.19 Initial marking M_0 of timed CPN model for state space analysis

The tokens on the places DelayTP and DelayTA describe the possible delays for transmission of packets on the network. We may have a short delay (25), a medium delay (50), or a long delay (75). The value bound to the variable `delay` is used in the time delay inscriptions of the two transitions to determine the delay when the packet is transmitted. As an example, consider the marking M_1 shown in Fig. 10.20. In this marking, there are six enabled bindings for the transition TransmitPacket:

⟨n=1, d="COL", success=true, delay=25⟩
⟨n=1, d="COL", success=true, delay=50⟩
⟨n=1, d="COL", success=true, delay=75⟩
⟨n=1, d="COL", success=false, delay=25⟩
⟨n=1, d="COL", success=false, delay=50⟩
⟨n=1, d="COL", success=false, delay=75⟩

This means that the node representing the marking M_1 in the state space will have six outgoing arcs, and the timed marking reached when TransmitPacket occurs depends only on the selected binding element, not on the value returned by a random number function as was the case when the function `Delay` was used to obtain the transmission delay. If we select the second of the above bindings to occur, we reach the marking shown in Fig. 10.21.

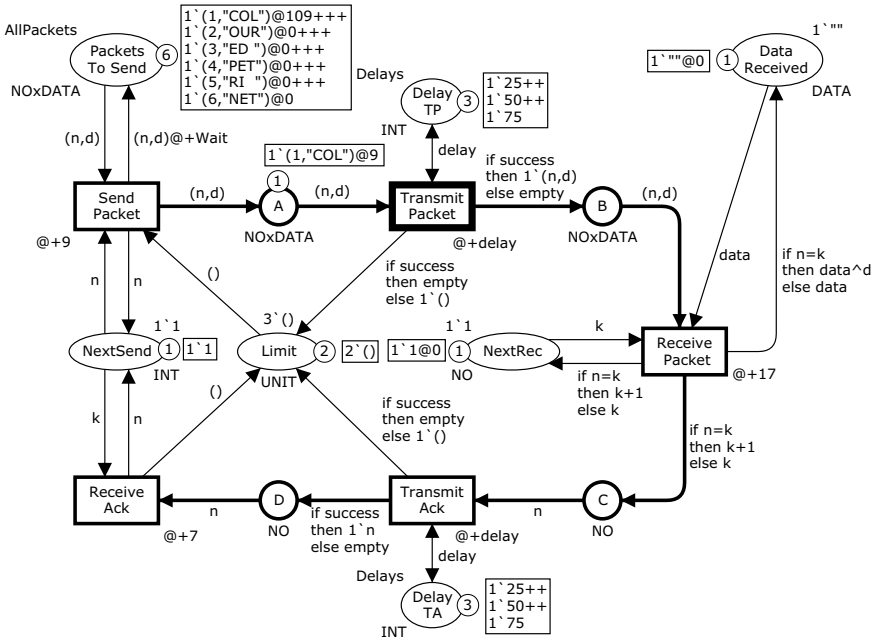

Fig. 10.20 Marking M_1 reached when SendPacket occurs at time 0 in M_0

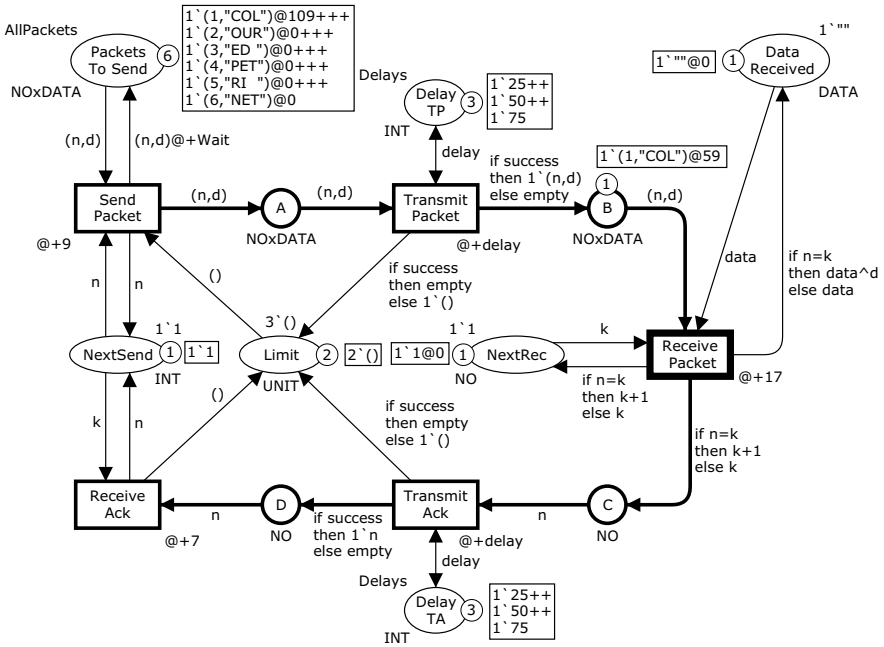

Fig. 10.21 Marking M_2 reached when TransmitPacket occurs at time 9 in M_1

The value of the global clock and the timestamps of the tokens are part of the timed marking. As an example, consider the marking M_1 in Fig. 10.20 and assume that a binding of TransmitPacket occurs in which the data packet is lost. We then reach the marking M_2^- shown in Fig. 10.22 in which the global clock has been increased to 109 at which time SendPacket is enabled corresponding to a retransmission of data packet 1. The initial marking M_0 in Fig. 10.22 and the marking M_2^- are *different* timed markings, since the values of the global clock are different in the two markings and the timestamps of one of the tokens are different. This means that these two markings will be represented by two different nodes in the timed state space.

Figure 10.23 shows an initial fragment of the state space for the timed protocol consisting of the markings reachable by the occurrence of at most three binding elements. Node 1 represents the initial marking, and the box next to each node gives information about the tokens on the individual places in the marking represented by the node. We have listed only places with a non-empty marking, and the places PacketsToSend, DelayTP, and DelayTA have been omitted since the colours of the tokens on these places do not change (except for the timestamps on PacketsToSend). The integer following the Time entry specifies the value of the global clock when the marking was created. For the labels on the arcs, we have used the same shorthand notation as in Chap. 7, except that for the transition TransmitPacket we have also specified the binding of the variable `delay`; for example, an arc labelled TPi+:50

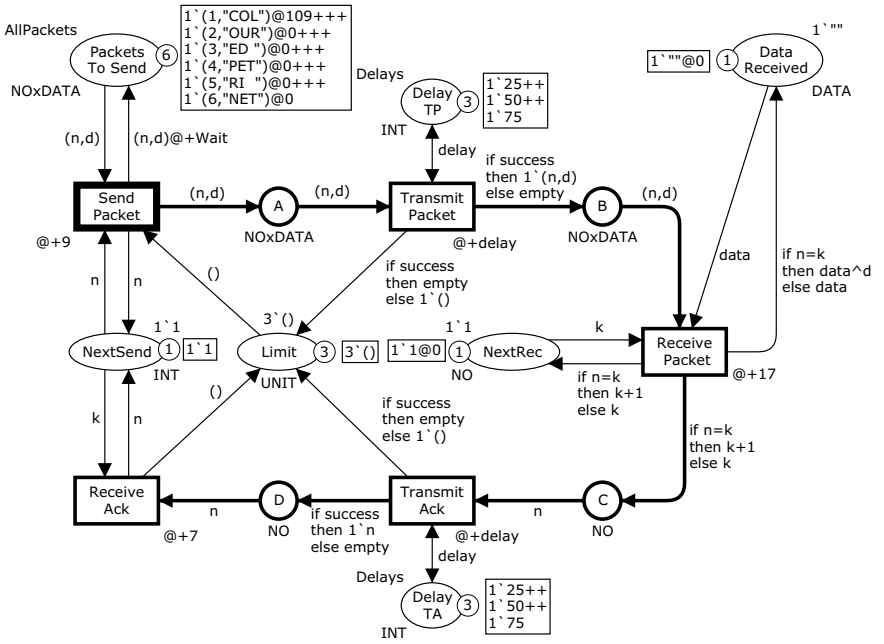

Fig. 10.22 Marking M_2^- reached after loss of data packet 1 in M_1

denotes an occurrence of TransmitPacket with a binding corresponding to a successful transmission of packet i where `delay` is bound to 50. There are three arcs leading from node 2 to node 4, all corresponding to an unsuccessful transmission of the first data packet. The difference between the three corresponding binding elements is the value bound to the variable `delay`. These arcs all lead to node 4, since the value bound to the variable `delay` does not matter when the occurrence of TransmitPacket corresponds to an unsuccessful transmission. The occurrences of the binding elements TP+1:25, TP1+:50, and TP1+:75 in node 2 lead to three different markings (represented by nodes 5, 3, and 6, respectively), since the timestamps on the token on place B in the resulting markings differ.

We may continue to lose the first data packet, and hence the timed state space can be infinite even if the corresponding untimed CPN model has a finite state space. To obtain a finite state space, we must set an upper bound on the value of the global clock. This limits the applicability of full state spaces for timed CPN models, but it is still possible to generate parts of timed state spaces and verify time-bounded properties such as checking whether a certain packet has been received before time 1000. Table 10.1 gives statistics for the numbers of nodes and arcs in partial state spaces obtained by not calculating successors for nodes when the global clock is greater than a certain 'Clock' value.

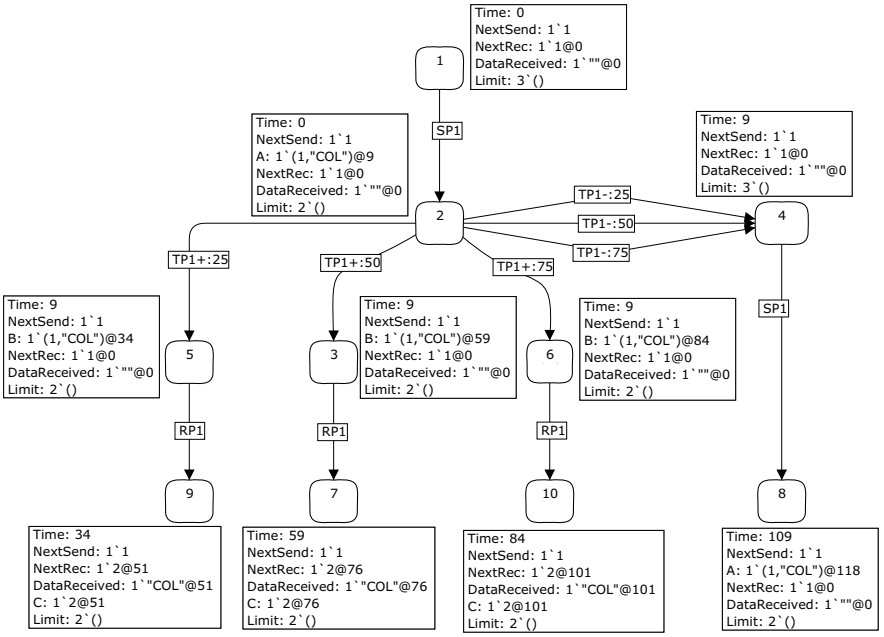

Fig. 10.23 Initial fragment of the state space for the timed protocol

Table 10.1 Size of partial state space for different time bounds

Clock	Nodes	Arcs	Clock	Nodes	Arcs
10	12	19	60	3 005	4 583
20	48	87	70	7 822	12 154
30	156	269	80	17 996	28 002
40	397	644	90	49 928	79 224
50	814	1 273	100	103 377	165 798

10.4 Time Equivalence Method

As discussed above, one of the main limitations on the use of state spaces for timed CPN models is that they are infinite for models containing cyclic behaviour. The problem is that the absolute notion of time as represented by the global clock and the timestamps of tokens is carried over into the timed markings of the state space.

Our protocol system contains cyclic behaviour since, for example, it is possible to keep losing and retransmitting the first data packet. As an example, consider

Figs 10.24 and 10.25, which show the timed markings M_1 and M_2 of the timed protocol system described in Sect. 10.3. Figure 10.24 shows the marking reached when SendPacket has occurred in the initial marking, and Fig. 10.25 shows the marking reached when the first data packet is lost and then retransmitted. The two markings are similar: the only difference is that the global clock has been advanced 109 time units and so has the timestamp of the token `(1,"COL")` on PacketsToSend. The two markings are represented by two nodes in the state space because the timestamps of the tokens and the values of the global clock differ.

The *time equivalence method* [19] has been developed to overcome this problem, and uses equivalence classes as introduced in Sect. 8.4 to factor out the absolute notion of time. This is done by replacing the absolute values in the global clock and the timestamps with relative values to construct a *condensed state space*. It can be proved that the condensed state space is finite provided that the state space of the corresponding untimed CPN model is finite. Furthermore, the condensed state space is constructed in such a way that all behavioural properties of the model that can be verified using the full state space can also be verified using the condensed state space.

The basic idea is to consider markings such as M_1 and M_2 to be equivalent and to compute a canonical representative for each equivalence class as follows:

- All timestamps which are less than the current model time are set to zero (they cannot influence enabling).

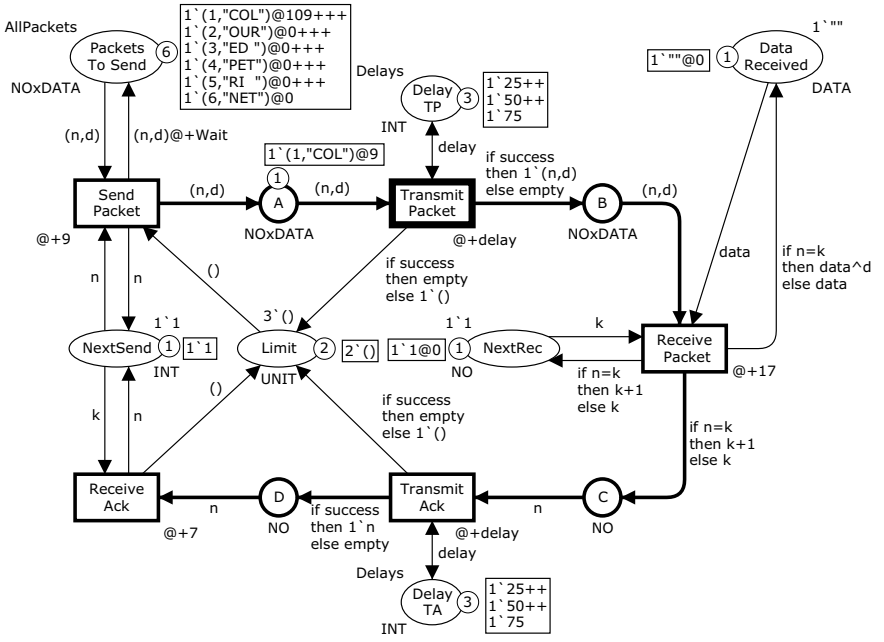

Fig. 10.24 Timed marking M_1 with an enabled transition at time 9

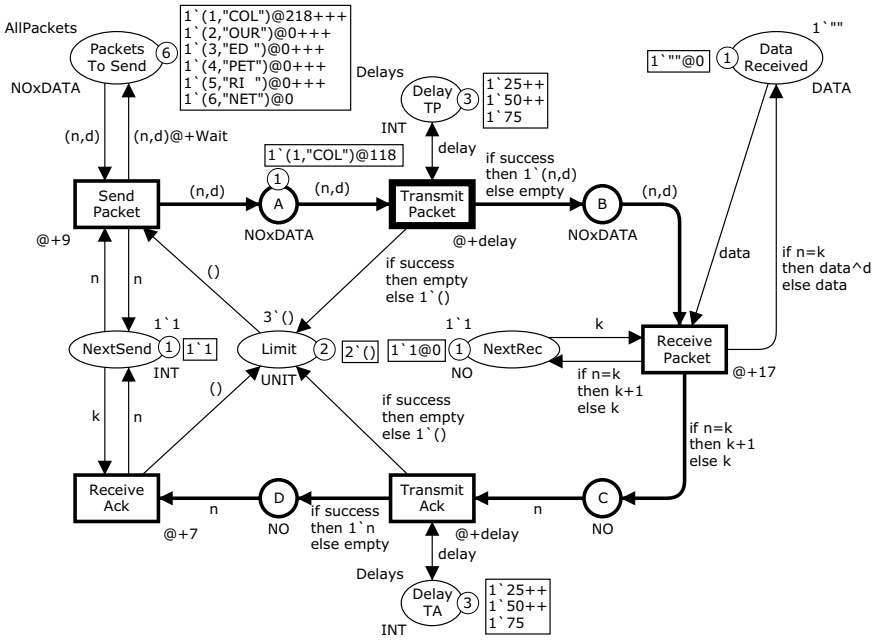

Fig. 10.25 Timed marking M_2 with an enabled transition at time 118

- The current model time is subtracted from all timestamps which are greater than or equal to the current model time.
- The current model time is set to zero.

Applying the above three rules to the markings in Figs 10.24 and 10.25 yields in both cases the canonical timed marking shown in Fig. 10.26. The value of the global clock is now 0. The timestamp of the token on A is also 0, and the timestamp of the token `(1,"COL")` on PacketsToSend is 100 since the value of the global clock has been subtracted from the timestamps of the corresponding tokens in the original markings. The transition TransmitPacket is enabled in the canonical marking and there are still 100 time units until the first data packet can be retransmitted, which was also the case in the original markings. Hence, the same occurrence sequences are possible in the canonical marking as in the original markings, but we have removed the absolute time. A formalisation of the above equivalence can be found in [19], including a proof that it is consistent for all timed CPN models.

The condensed state space for a timed CPN model can be computed fully automatically. The consistency of the equivalence has been proven once and for all [19] and the user does not have to provide any predicate for expressing the time equivalence, because it has been implemented in CPN Tools once and for all, for all CPN models. It has been shown [19] that all properties of the system expressible in the real-time temporal logic RCCTL* [38] are preserved in the condensed state space. This set of properties includes all of the standard behavioural properties of

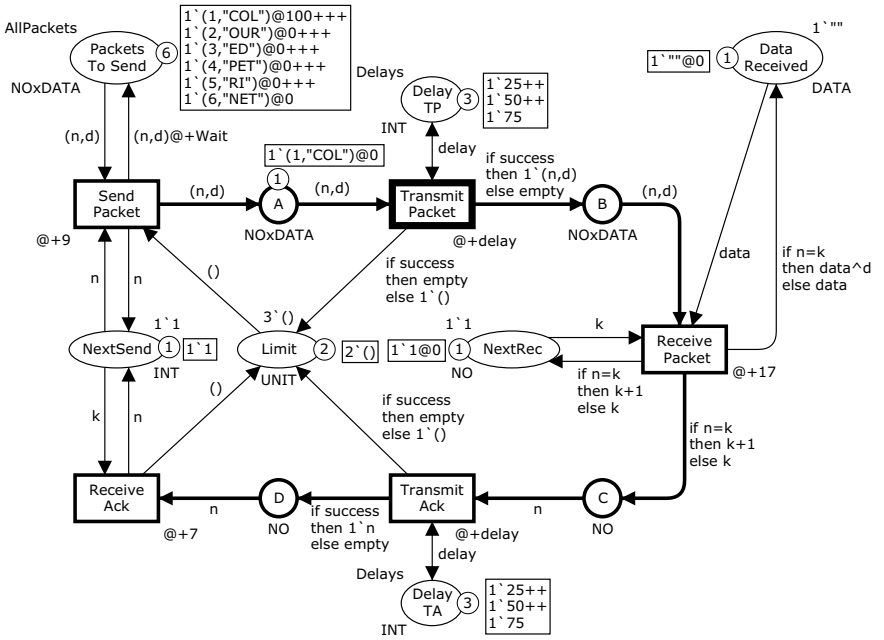

Fig. 10.26 Canonical timed marking for M_1 and M_2

CPN models discussed in Chap. 7. Table 10.2 shows some statistics for the size of the condensed state space for the protocol system. The time equivalence method has also been used in the industrial application described in Sect. 14.3.

Table 10.2 Statistics for application of the time equivalence method

Limit	Packets	Nodes	Limit	Packets	Nodes
1	10	81	5	2	88 392
1	20	161	5	4	308 727
1	50	401	7	1	13 198
1	100	801	7	2	145 926
2	5	3 056	7	3	323 129
2	10	6 706	10	1	20 062
2	20	14 006	10	2	244 990
2	50	35 906	12	1	24 630
3	1	2 699	12	2	335 651
3	5	85 088	13	1	26 914
3	15	306 118	13	2	391 743