Timed Protocol

Abstract
This example shows how the CP-net from "Simple Protocol" can be turned into a timed CP-net. The timed CP-net specifies how long time the individual operations take and how long time the sender should wait before it makes a retransmission. The timed CP-net can be used to experiment with different waiting times to determine which one is the best – in the sense that it transmits the message fast without using the network too much (i.e. without making too many retransmissions).

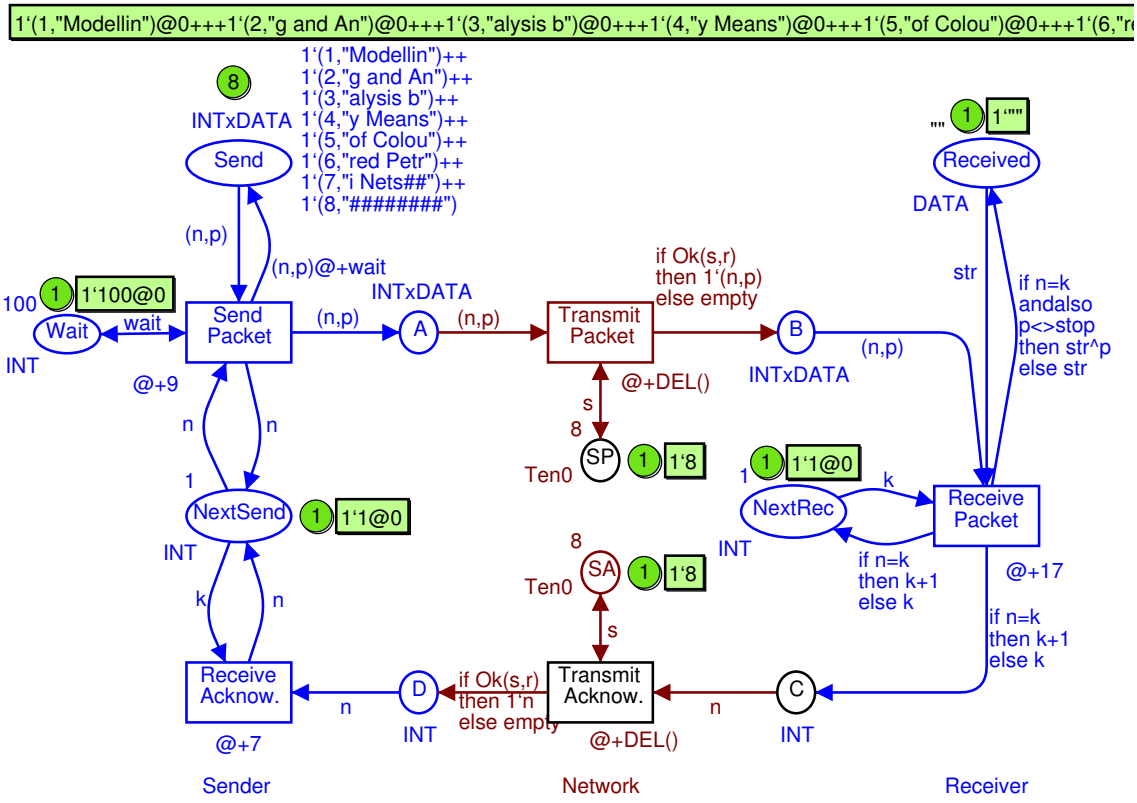
The example is taken from Sect. 5.5 of Vol. 2 of the CPN book.

Developed and Maintained by:
Kurt Jensen, Aarhus University, Denmark (kjensen@daimi.au.dk).

CPN Model

This example explains how the CP-net from "Simple Protocol" can be turned into the timed CP-net shown below. The timed net has the same net structure as the untimed net – except that a new place *Wait* has been added (in the left-hand side). This place is used to specify how long time the transition *Send Packet* should wait before retransmitting a packet..

From the declarations it can be seen that the colour set *INT* is timed while




```
color INT = int timed;
color DATA = string;
color INTxDATA= product INT*DATA;
var n,k, wait: INT;
var p, str: DATA;
val stop = ”########”;

color Ten0 = int with 0..10;
color Ten1 = int with 1..10;
var s: Ten0; var r: Ten1;
fun Ok(s:Ten0, r:Ten1) = (r<=s);

color NetDelay = int with 25..27;
fun DEL() = NetDelay.ran();
```

DATA, *Ten0* and *Ten1* are not. By the convention, the structured colour set *INTxDATA* is timed – because it contains the timed component *INT*.

We have added a time region (starting with @+) to each of the five transitions. Intuitively, the time region describes how long time the corresponding operation takes. Now let us take a closer look at the five different transitions in the protocol system.

Send Packet has a time region: @+9. This implies that a common delay of 9 time units is added to the time stamps of *all* output tokens. The tokens created at *A* and *Next Send* get a time stamp that is the current time r* (at which the transition occurs) plus 9. The output arc to place *Send* specifies an additional time delay to be used for the tokens added to *Send*. This token will get a time stamp which is r*+9+100 (since the variable *wait* is bound to 100). Intuitively, the delay 9 represents the time used to send a packet, while the delay 100 represents the time that has to elapse before a retransmission, i.e., before *Send Packet* occurs once more for the same packet. A retransmission will only happen if the number in *Next Send* remains unaltered for 9+100 time units, i.e., if no acknowledgement for the packet is received inside this time period.

Transmit Packet has a time region: @+*DEL()*, where *DEL* is declared to be an ML function returning a random element from the colour set *NetDelay* (i.e., a random integer in the interval between 25 and 75). This implies that the duration of a transmit operation may vary inside this interval.

Receive Packet and *Receive Acknowledgement* have time regions specifying a fixed duration (17 and 7, respectively), while *Transmit Acknowledgement* has a variable duration time, between 25 and 75. All time delays are specified by means of an ML expression. Hence, it is easy to use statistical functions specifying more complex types of delays (e.g., exponential distributions).

Note that the token in *Next Send* has a time stamp. Intuitively, this means that the sender can not start a new *Send Packet* or a new *Receive Acknowledgement* as long as one of these operations is already ongoing. If the *Sender* has multiple processes (threads), allowing an unlimited number of *Sender* operations to be performed at the same time, we simply make the colour set of *Next Send* untimed. A similar remark applies for the operations of the *Receiver* and the colour set of *Next Rec*.

After a number of simulation steps the timed CP-net may reach a dead final marking with the contents shown below. From the time stamp of *Next Rec* it can be seen that the last packet was received at time 1194. Analogously, the time stamp at *Next Send* tells us that the last acknowledgment was received at time 1269. The time stamps at place *Send* tell us the times at which the individual packets would have been retransmitted (had this become necessary). As an example, we can see that the first packet would have been retransmitted at time 218, the second at time 234, the third at 324, and so on.

By means of our timed CPN model we can investigate the performance of the protocol, e.g., experiment with different values for the retransmission delay specified by *Wait*. A short delay increases the chance of making unnecessary retransmissions. It also increases the chance that a *Receive Acknowledgement* operation is postponed, because the *Sender* process is engaged in a retransmission. A long delay means it may take too long before the *Sender* recognises that a packet or an acknowledgement has been lost. By making a number of simulations, with different values at *Wait*, we can determine the optimal value for the retransmission delay.