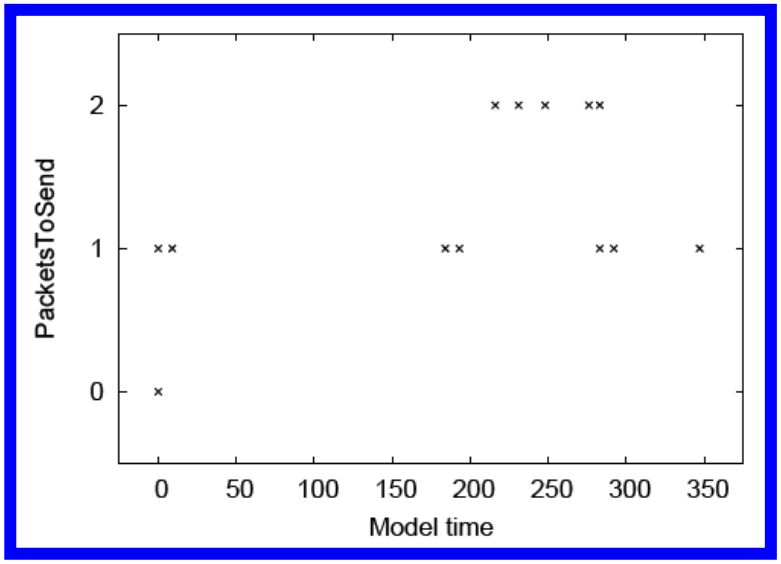
Coloured Petri Nets

Modelling and Validation of Concurrent Systems

Chapter 12:
Simulation-based Performance Analysis

Kurt Jensen &
Lars Michael Kristensen

{kjensen,lmkristensen}
@daimi.au.dk



Performance Analysis

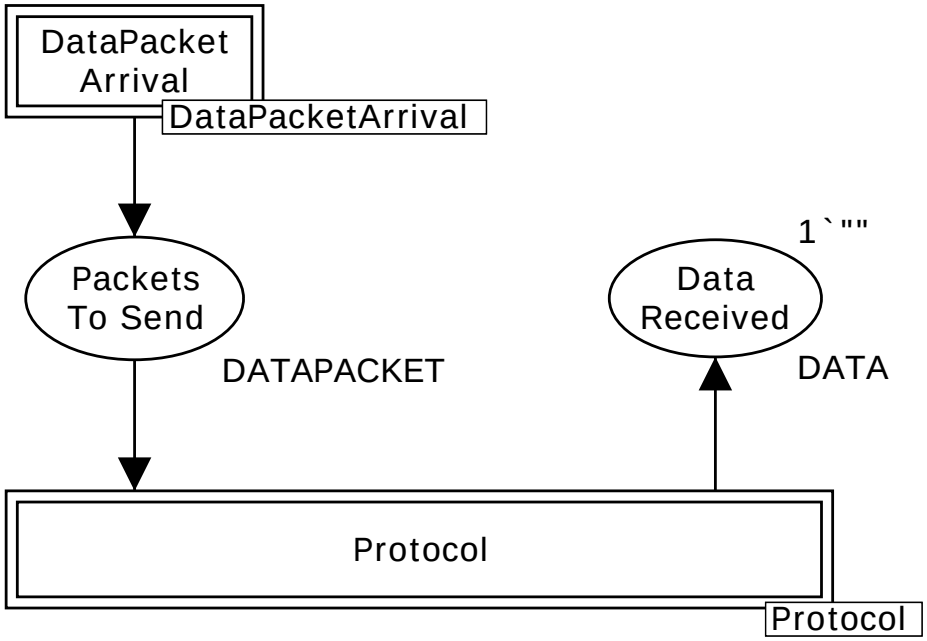
- Analysing performance is a central issue in the development and configuration of concurrent systems:
  - Evaluate existing or planned systems.
  - Compare alternative implementations of a system.
  - Search for optimal configuration(s) of a system.

- Performance measures of interests include average queue lengths, average delay, throughput, and resource utilisation.

- Performance analysis of timed CPN models is based on conducted a set of lengthy simulations of the CPN model.
- Numerical data is collected from the occurring binding elements and markings reached to estimate the performance measures.

Timed Protocol for Performance Analysis

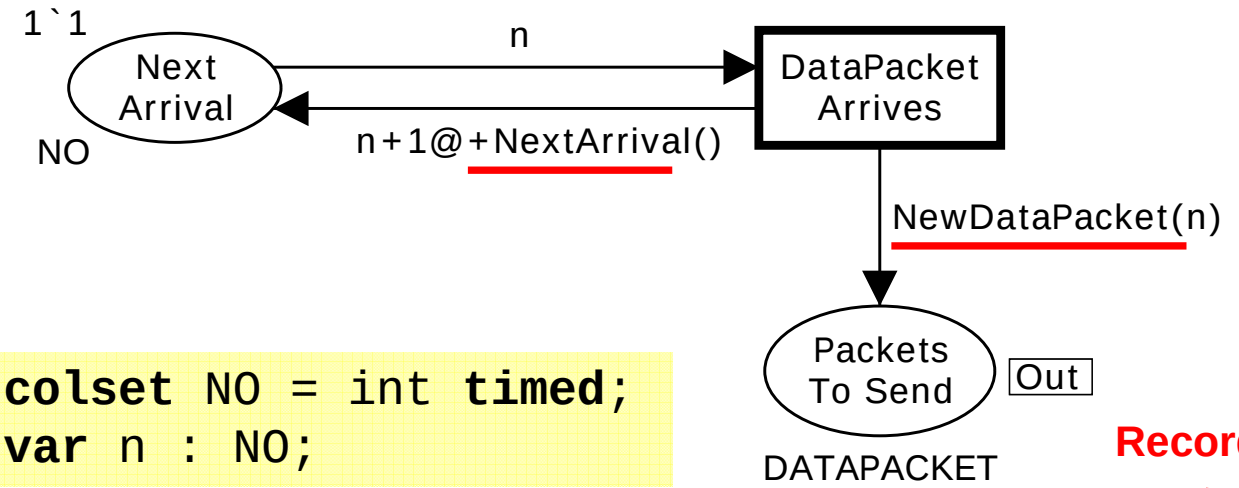
- Modifications of the timed CPN model required to properly estimate performance measures of interests:



- Arrival of data packets to be transmitted (workload).
- Recording data packet arrival time to measure delay.
- Probability of packet loss on the network.

Arrival of Data Packets

- Creation of data packets during simulation of the CPN model:




```
colset NO = int timed;
var n : NO;
```

```
colset DATA       = string timed;
colset TOA        = int;
colset DATAPACKET = product NO * DATA * TOA timed;
```

Record Time Of Arrival

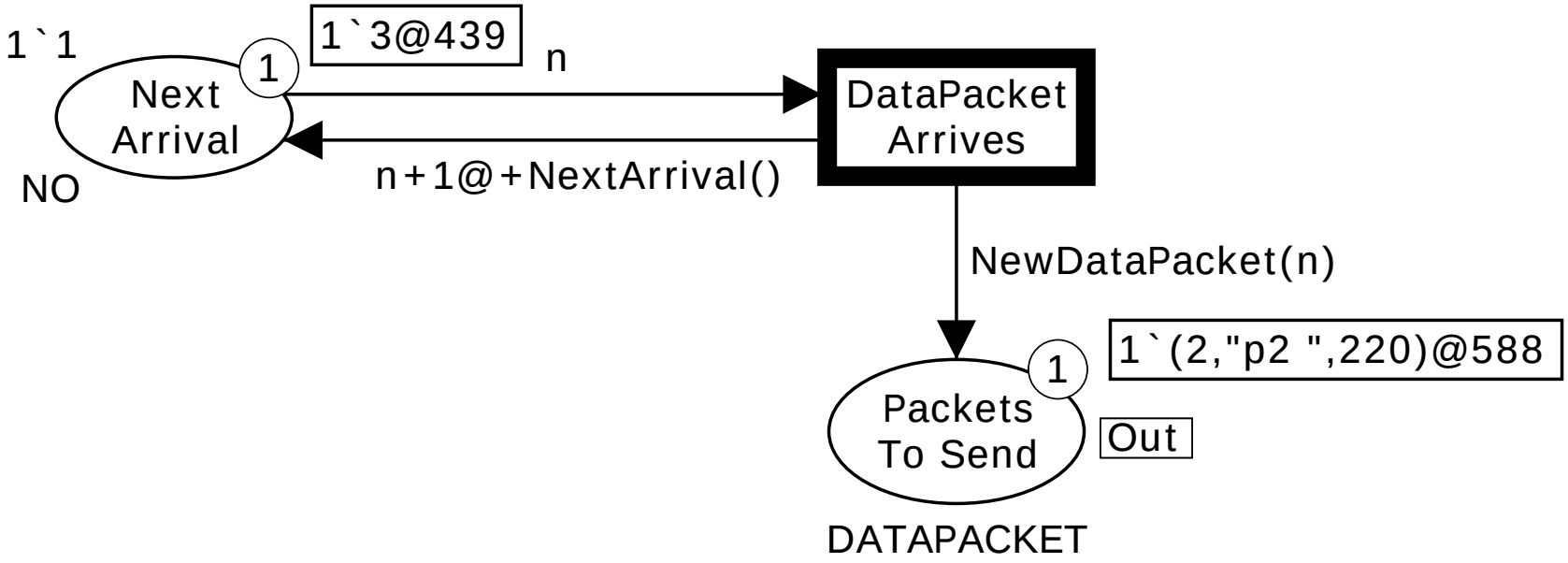
Returns value of global clock

```
fun NextArrival() = discrete(200,220)
```

Uniform discrete distribution

```
fun NewDataPacket n = (n, "p"^NO.mkstr(n)^" ", ModelTime())
```

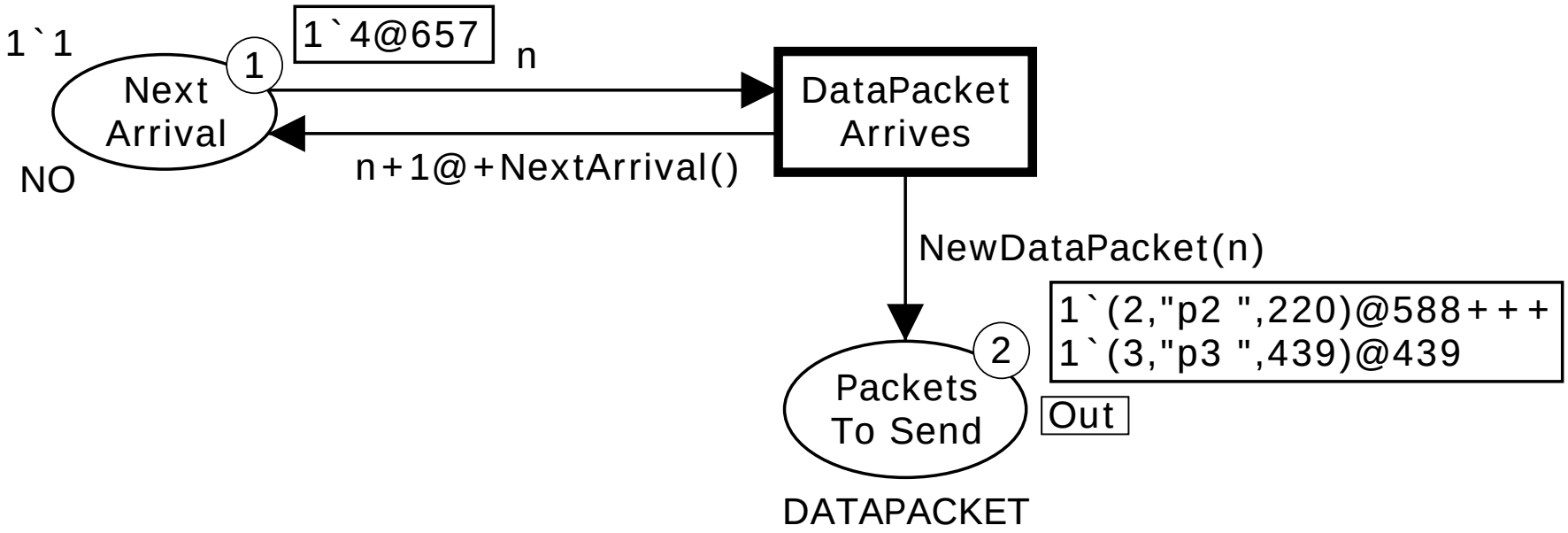
Example: Data Packet Arrival M_1



(DataPacketArrives, <n=3>) **439**

```
NextArrival() = 218
```

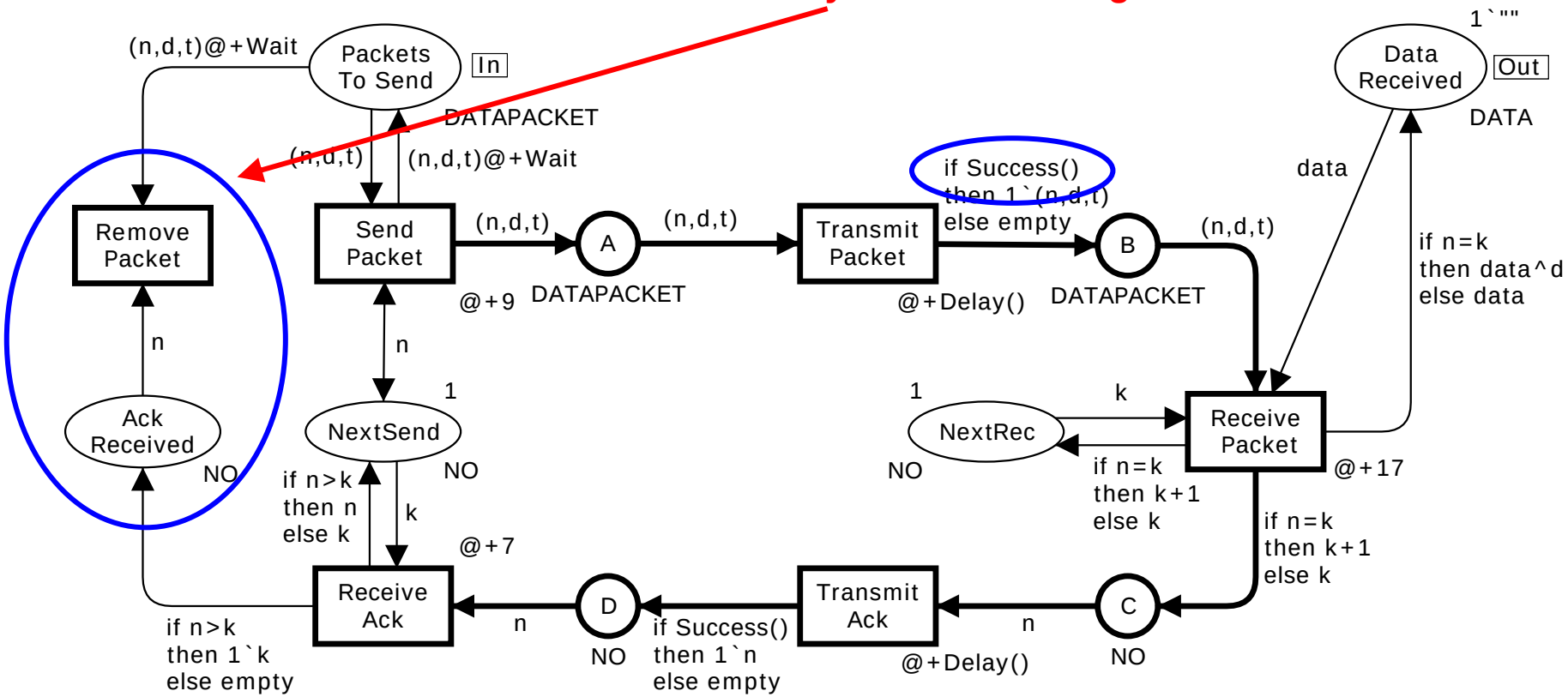
Data Packet Arrival M_2



- Next data packet will arrive at time 657 = 439 + 218.

Protocol Module

Data packets are removed as they are acknowledged

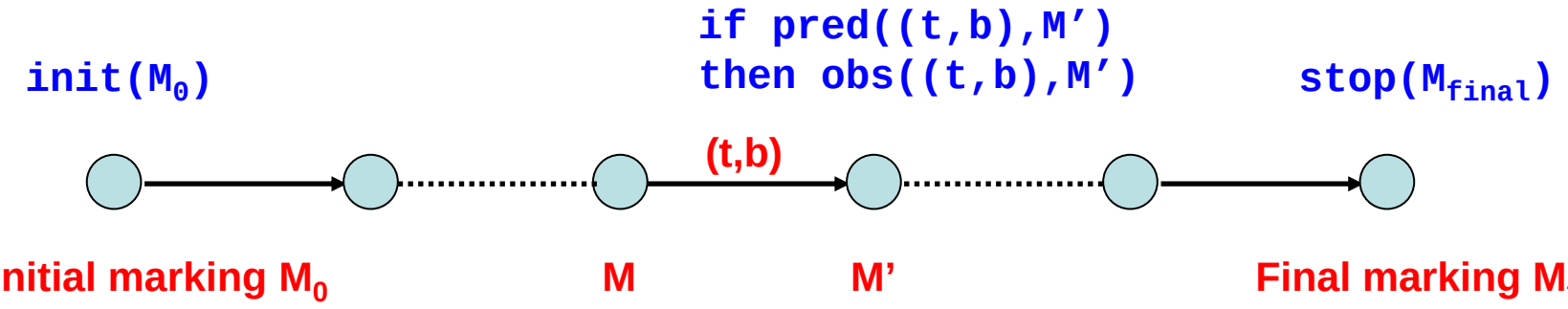


Uniform continuous distribution

```
val successrate = 0.9
fun Success() = uniform(0.0,1.0) <= successrate;
```

Data Collection and Monitors

- Data collection in CPN Tools is supported by data collector monitors to be defined for the performance measures of interest.
- Data collectors monitors extract numerical data from occurring binding elements and the markings reached in a simulation.
- A data collector is defined by means of its monitoring functions:
  - Initialisation function: collects data from the initial marking.
  - Predicate function: determines when data is collected.
  - Observation function: determines what data is collected.
  - Stop function: collects data from the final marking.



Monitoring Functions

- The monitoring functions are implemented in CPN ML and each consists typically of 5-10 lines of code.

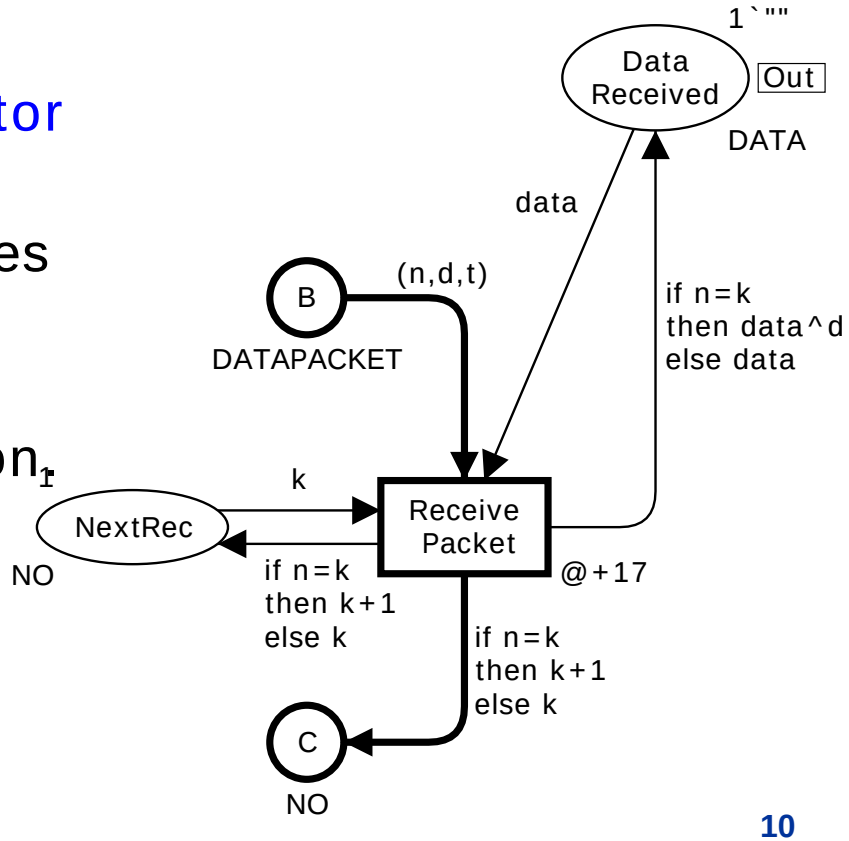
- CPN Tools supports a set of standard data collection monitors for which the monitoring functions can be generated automatically.
- CPN Tools can generate template code for user-defined data collector monitors which can then be adapted by the user.

- A monitor has an associated set of places and transitions determining what can be referred to in monitoring functions.
- This is exploited by CPN Tools together with locality to reduce the number of times that the monitoring functions are invoked.

Data Packets Received

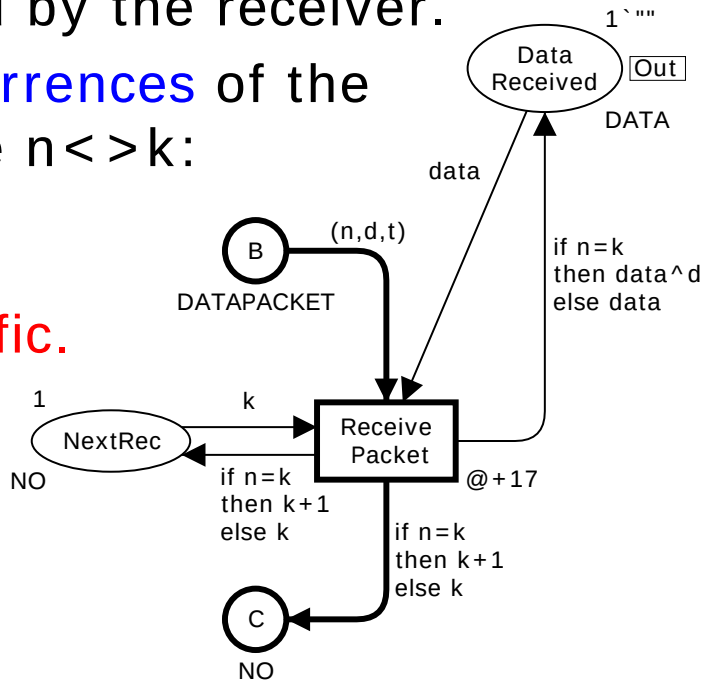
- Number of data packets being processed by the receiver.
- Can be estimated by counting the number of occurrences of the ReceivePacket transition:

- A count transition occurrences monitor can be used for this purpose.
- A counter within the monitor indicates the number of occurrences.
- A standard data collection monitor: user only have to select the transition.



Duplicate Data Packets Received

- Number of duplicate data packets received by the receiver.
- Estimated by counting the number of occurrences of the ReceivePacket transition in bindings where n<>k:
- A user-defined data collector monitor is required since the property is model-specific.
- Predicate function **pred**:




```
fun pred (Protocol'Receive_Packet
                (1,{d,data,k,n,t})) = true
  | pred _ = false
```

- Observation function **obs**:

```
fun obs (Protocol'Receive_Packet (1,{d,data,k,n,t})) =
                  if n<>k then 1 else 0
  | obs _ = 0
```

Data Packet Delay

- Time that elapses from when a data packet arrives for transmission on place PacketsToSend until it is received on DataReceived.
- Can be estimated based on the time of arrival information put into data packets when they arrive for transmission:

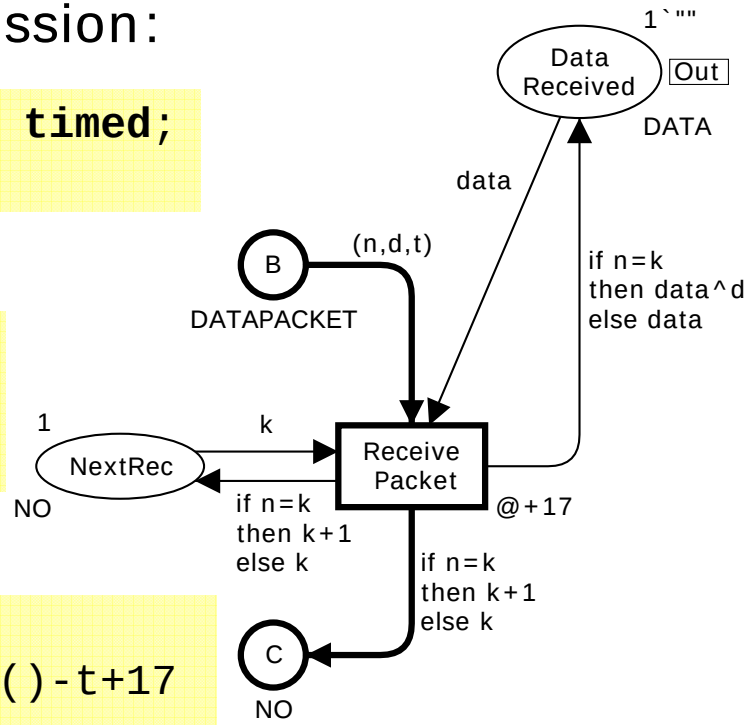
```
colset DATAPACKET = product NO * DATA * TOA timed;
var t : TOA;
```

- Predicate function **pred**:

```
fun pred (Protocol'Receive_Packet
                  (1,{d,data,k,n,t})) = n=k
  | pred _ = false
```

- Observation function **obs**:

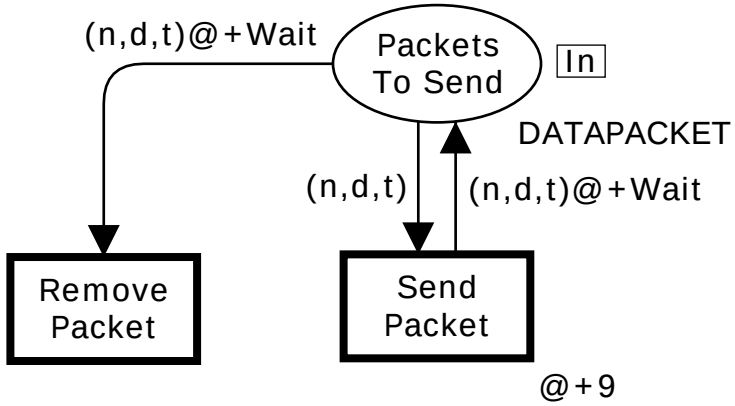
```
fun obs (Protocol'Receive_Packet
            (1, {d,data,k,n,t})) = ModelTime()-t+17
 |  obs _ = 0
```



Queue of Data Packets to Send

- Number of packets in queue at the sender.
- Can be estimated by counting the number of tokens on PacketsToSend:

- A marking size monitor can be used for this purpose.
- A standard data collection monitor: user only have to select the place.

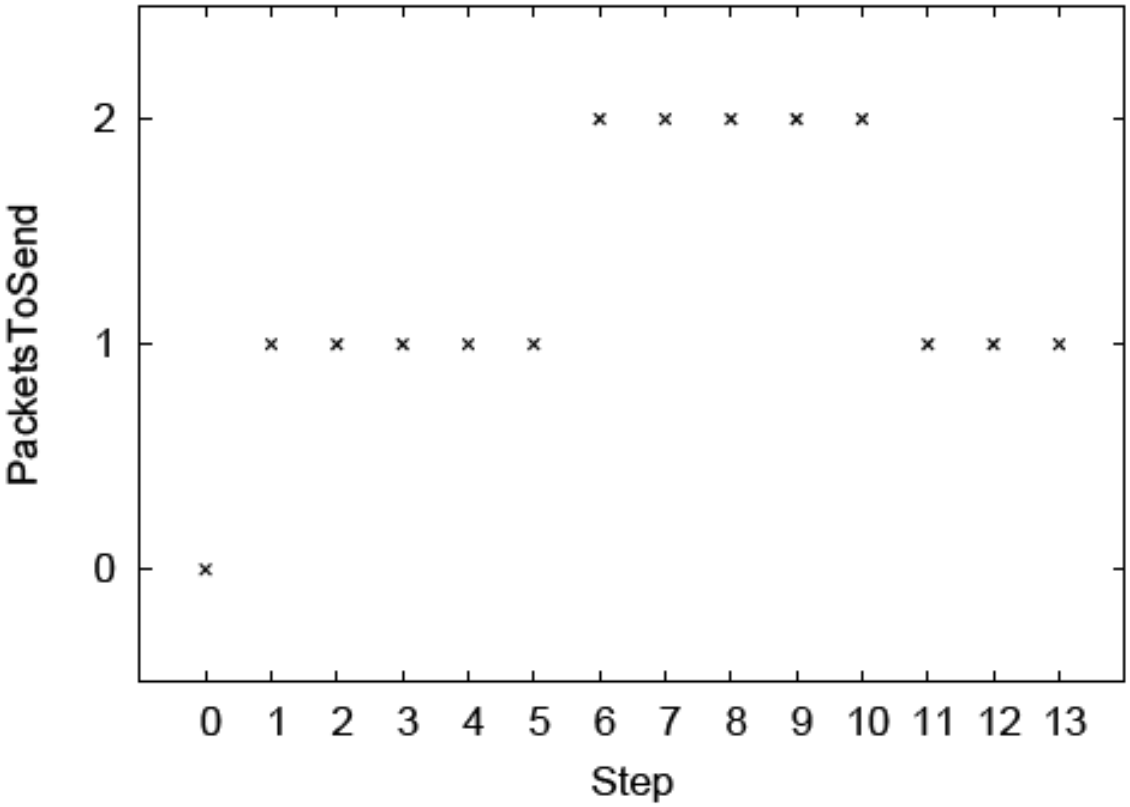


- The marking size monitor for PacketsToSend takes into account the amount of time that tokens are present on the place.

Example Simulation

Step	Time	Binding element	Tokens
–	0	–	0
1	0	(DataPacketArrives, ⟨n=1⟩)	1
2	0	(SendPacket, ⟨n=1, d="p1", t=220⟩)	1
3	9	(TransmitPacket, ⟨n=1, d="p1", t=220⟩)	1
4	184	(SendPacket, ⟨n=1, d="p1", t=220⟩)	1
5	193	(TransmitPacket, ⟨n=1, d="p1", t=220⟩)	1
6	216	(DataPacketArrives, ⟨n=2⟩)	2
7	231	(ReceivePacket, ⟨n=1, d="p1", t=220, data="", k=1⟩)	2
8	248	(TransmitAck, ⟨n=2, t=220⟩)	2
9	276	(ReceiveAck, ⟨n=2, t=220,k=2⟩)	2
10	283	(SendPacket, ⟨n=2,d="p2 ",t=436⟩)	2
11	283	(RemovePacket, ⟨n=1, t=220,d="p1 "⟩)	1
12	292	(TransmitPacket, ⟨n=2,d="p2 ", t=436⟩)	1
13	347	(ReceivePacket,⟨n=2,k=2,d="p2 ",t=436, data="p1 "⟩)	1

Discrete-parameters statistics

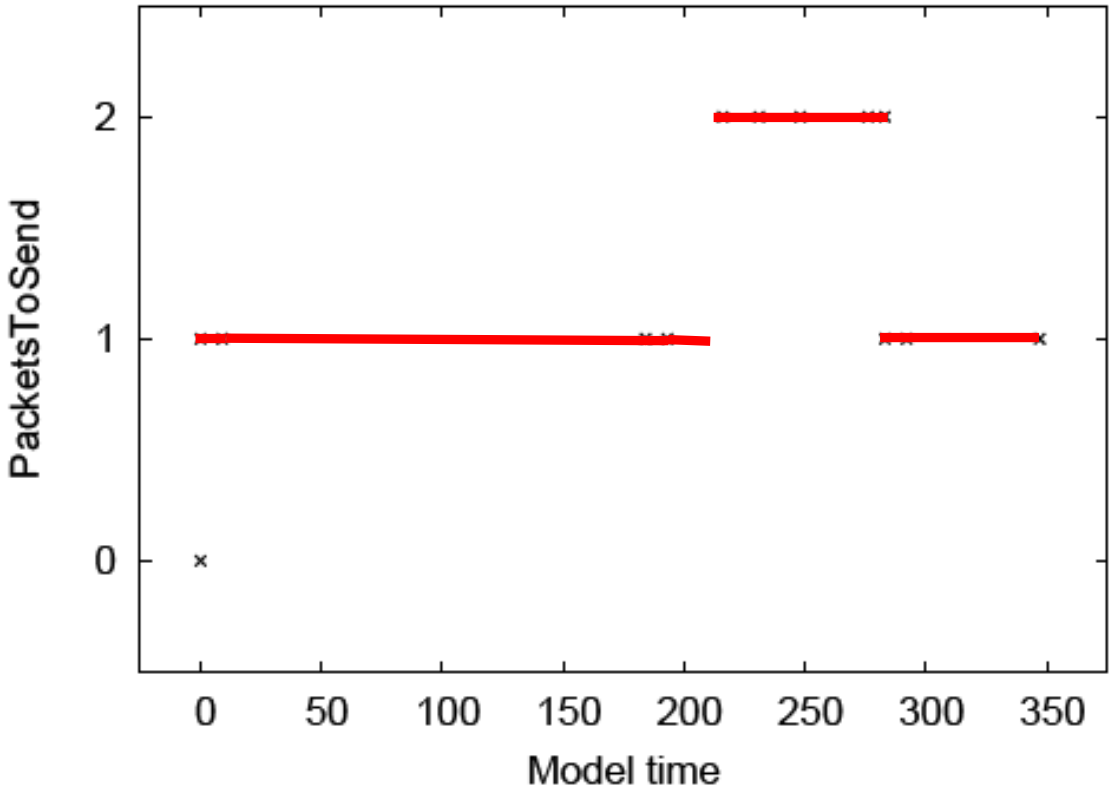


- Computing the average number of tokens:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\frac{0+1+1+1+1+1+2+2+2+2+2+1+1+1}{14} = 1.26$$

Continuous-time statistics



- Time-average number of tokens:

$$sum_t = (\sum_{i=1}^{n-1} x_i(t_{i+1} - t_i)) + x_n(t - t_n)$$

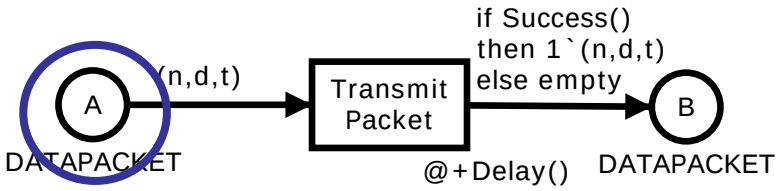
$$avrg_t = \frac{sum_t}{t - t_1}$$

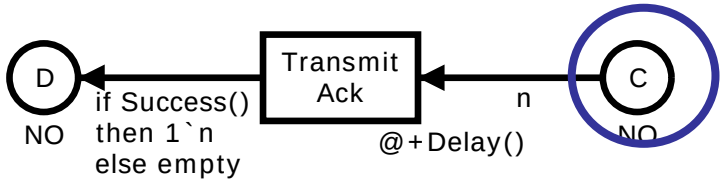
$avrg_{347}$ = 414/347 = 1.19

$$\begin{aligned} &0\times0 + 1\times0 + 1\times9 + 1\times175 + 1\times9 + 1\times23 + 2\times15 + \\ &2\times17 + 2\times28 + 2\times7 + 2\times0 + 1\times9 + 1\times55 + 1\times0 = 414 \end{aligned}$$

Network buffer queues

- Total number of data packets and acknowledgement in queue for transmission on the network.
- Count the sum of the number of tokens on the places A and C:





- Requires a user-defined data collector monitor.

Network buffer queue

- Predicate function returns true whenever one of the transitions SendPacket, TransmitPacket, ReceivePacket or TransmitAck occurs.

- Initialisation function **`init`**:

data collection in initial marking is optional

```
fun init (Protocol'A_1_mark : DATAPACKET tms,
          Protocol'C_1_mark : ACK tms) =
   SOME ((size Protocol'A_1_mark) + (size Protocol'C_1_mark))
```

- Observation function **`obs`**:

```
fun obs (bindelem, Protocol'A_1_mark : DATAPACKET tms,
                   Protocol'C_1_mark : ACK tms) =
    (size Protocol'A_1_mark) + (size Protocol'C_1_mark)
```

Throughput

- Number of non-duplicate data packets delivered by the protocol per time unit.
- Can be estimated at the end of a simulation by dividing the data packets received by the model time in the final marking.

- A data collector monitor with a stop function can be used for this purpose:

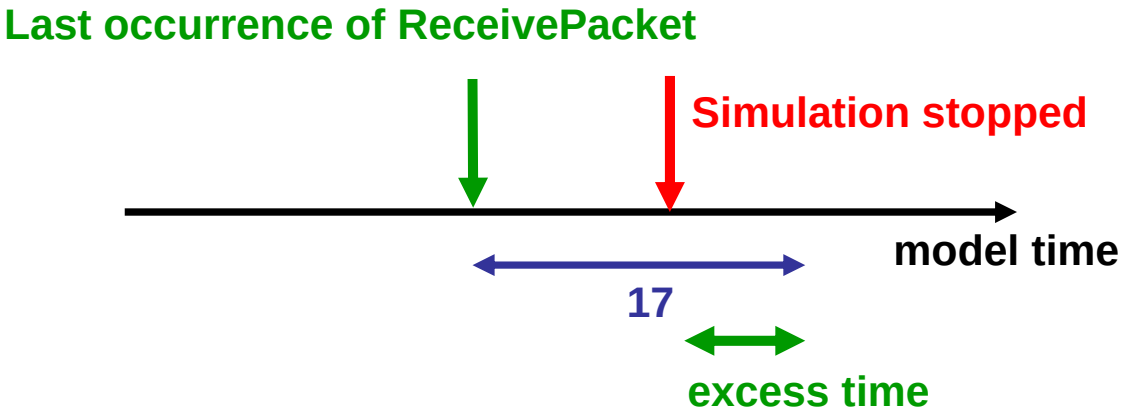
Observations made by the data packet delay monitor

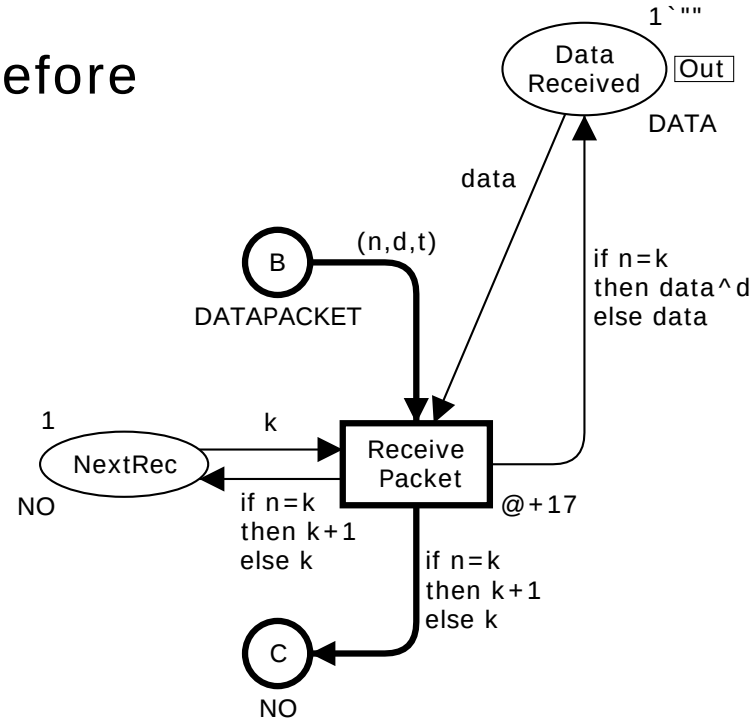
```
fun stop () =
  let
    val received  = DataPacketDelay.count()
    val modeltime = ModelTime()
  in
    SOME ((Real.fromInt received)/(Real.fromInt(modeltime))
  end
```

Receiver utilisation

- Proportion of time that the receiver is busy processing packets.
- Can be computed by considering the number of occurrences of the ReceivePacket transition which each takes 17 units of model time.

- Simulation may have been stopped before received operation has ended:





Received Utilisation – stop function

```
fun stop (Protocol'NextRec_1_mark : NO tms) =
  let
    val ts          = timestamp (Protocol'NextRec_1_mark)
    val busytime    = DataPacketReceptions.count() * 17

    val modeltime   = ModelTime();
    val excesstime = Int.max(ts - modeltime,0)

    val busytime'   =  busytime - excesstime
  in
    SOME ((Real.fromInt busytime')/(Real.fromInt modeltime))
  end
```

Performance report

- Statistics for the data collector monitor can be saved in a performance report.
- Example: Simulation with 10,000 steps, model time 272,658 and 1298 data packets received:

- Continuous-time statistics:

Monitor	Count	Average	StD	Min	Max
PacketsToSendQueue	4,206	0.5124	0.6998	0	5
NetworkBufferQueue	3,220	0.0531	0.2242	0	1

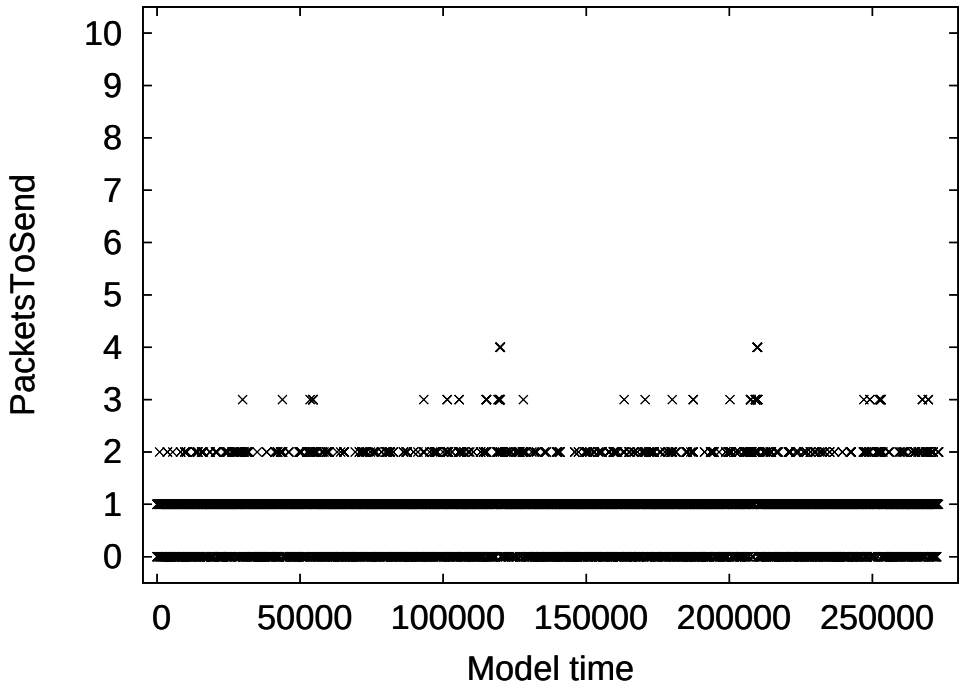
- Discrete-parameters statistics:

Monitor	Count	Sum	Average	StD	Min	Max
DataPacketReceptions	1,445	1,445	1.0	0.0	1	1
DuplicateReceptions	1,445	147	0.1017	0.3024	0	1
DataPacketDelay	1,298	93,976	72.4006	94.3382	31	800
Throughput	1	0.0048	0.0048	0.0	0.0048	0.0048
ReceiverUtilization	1	0.0901	0.0901	0.0	0.0901	0.0901

Simulation output

- The raw numerical data collected by the monitors is also saved in data collector log files:

#data	counter	step	time
0	1	0	0
1	2	1	0
1	3	2	0
0	4	7	43
1	5	8	211
1	6	9	211
0	7	14	254
1	8	15	417



Model Parameters

- When conducting performance analysis it is often of interest to investigate different configurations of the modelled system.
- This requires the parameters of the CPN model to be changed between simulation runs.
- Parameters for the protocol system:

```
val successrate = 0.5;
val Wait = 175;

fun NextArrival() = discrete(200,220)
fun Delay()       = discrete(25,75)
```

- Changing parameters requires a partial syntax check and regeneration of simulation code between simulation runs.

Reference variables

- Reference variables allow parameters to be changed without requiring syntax check and simulation code generation.

```
globref successrate    = 0.9;
globref packetarrival  = (200,220);
globref packetdelay    = (25,75);
globref retransmitwait = 175;
```

- Functions are used in the model inscriptions to access the parameters:

```
fun Success()     = uniform(0.0,1.0) <= !successrate;
fun Wait()        = !retransmitwait;
fun Delay()       = discrete(!packetdelay);
fun NextArrival() = discrete(!retransmitwait);
```

Revised Protocol Module