# Performance Analysis using Coloured Petri Nets

Lisa Wells
CPN Centre, Dept. of Computer Science, University of Aarhus
Åbogade 34, 8200 Århus N, Denmark
wells@daimi.au.dk

## Abstract

*This paper provides an overview of improved facilities for performance analysis using coloured Petri nets. Coloured Petri nets is a formal method that is well suited for modeling and analyzing large and complex systems. The paper describes steps that have been taken to make a distinction between modeling the behavior of a system and observing the behavior of a model. Performance-related facilities are discussed, including facilities for collecting data, running multiple simulations, generating statistically reliable simulation output, and comparing alternative system configurations.*

## 1 Introduction

A large body of research is concerned with using formal methods for performance analysis [1]. Most of this research focuses on the use of sophisticated methods for generating and solving analytical models, such as continuous-time Markov chains, for performance analysis. An advantage of using analytical models is that they can provide exact answers regarding the performance of a model. However, for even small configurations of a system it may be impossible to generate the analytical models needed for performance analysis due to the state explosion problem. Furthermore, it can be difficult to create accurate and understandable models of industrial-sized systems when using, e.g. low-level Petri nets.

Coloured Petri nets [9, 11] (CP-nets or CPN) is a formal method that is well suited for modeling and analyzing large and complex systems for several reasons: hierarchical models can be constructed, complex information can be represented in the token values and inscriptions of the models, and mature and well-tested tools exist for creating, simulating, and analyzing CPN models. In the past, simulation of CP-nets has primarily been used for debugging, validation, and checking of logical correctness.

CP-nets and the Design/CPN tool [3, 6] have been used to analyze many computer and telecommunication systems. Verification studies have analyzed the functionality of protocols, e.g. WAP [8] and RSVP [15]. A few performance studies have also been done, e.g. in the areas of web servers [16], ATM network algorithms [4], and bank transaction processing [2]. Design/CPN can be compared to tools such as GreatSPN and Ex*Spect* (see [14] for references and additional tools).

This paper provides an overview of improved facilities for simulation-based performance analysis using coloured Petri nets. This paper will focus on how CP-nets can be used to analyze network protocols, but the facilities discussed here can be used to analyze any kind of system. Network protocols are particularly interesting because it is often important to analyze both the functionality and the performance of a protocol. One of the strengths of CP-nets is that they can be used for both functional and performance analysis. A general-purpose simulation tool such as Arena [10] provides sophisticated and excellent support for analyzing the performance of many kinds of systems, including networks. However, it is virtually impossible to analyze the functionality of a system using such a simulation package.

The paper is structured as follows. Section 2 introduces a CPN model of a stop-and-wait protocol that will be used as an example throughout the paper. Section 3 discusses steps that have been taken to separate modeling the behavior of a system from observing the behavior of the model in Design/CPN. Section 4 presents the improved performance facilities for Design/CPN and related tools. Future work is discussed in Sect. 5.

## 2 Example: Stop-and-Wait Protocol

This section briefly introduces a CPN model of a stop-and-wait protocol from the data link control layer of the OSI network architecture. The protocol is quite simple, but it is sufficient for introducing the basic concepts related to using CP-nets for performance analysis. The following description of the model is taken from [11] which provides both a thorough description of the model and a practical introduction to CP-nets.

Proceedings of the 10th IEEE Int'l Symp. on Modeling, Analysis, & Simulation of Computer & Telecommunications Systems (MASCOTS'02)
1526-7539/02 $17.00 © 2002 IEEE

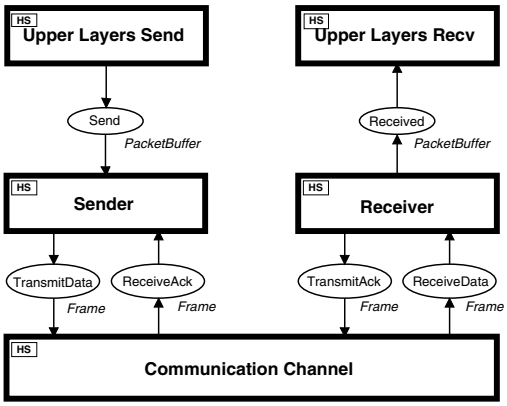


**Figure 1. Stop-and-wait CPN model.**

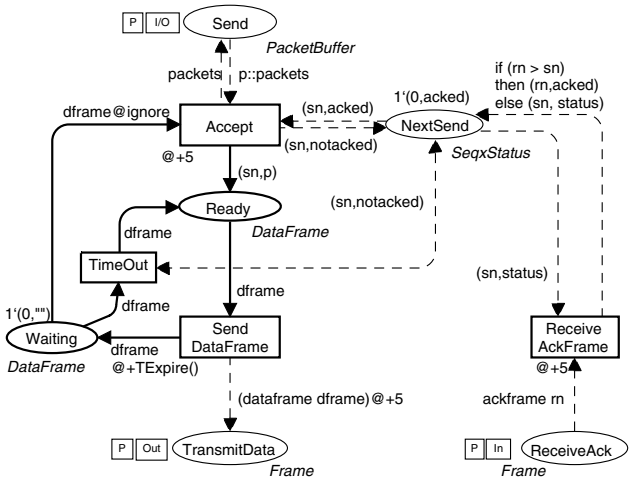


**Figure 2. Sender module.**

Figure 1 shows an overview of a timed, hierarchical CPN model of the stop-and-wait protocol. The system consists of a sender transmitting data packets to a receiver across an unreliable, bi-directional communication channel. The sender accepts data packets from protocols in the upper layers of the OSI network architecture. Similarly, the receiver passes packets that have been properly received to the upper layers of the protocol stack. The Upper Layers Send module generates packets on-the-fly during a simulation of this CPN model. The Communication Channel module provides a simple model of an unreliable network in which packet loss and overtaking can occur. The stop-and-wait protocol is modeled in detail in the Sender and Receiver modules.

Figure 2 shows the Sender part of the CPN model. The states of a CP-net are represented by a number of *tokens* positioned on *places*, which are drawn as ellipses. Each token carries a data value, such as an integer, a string, or a list of booleans. No tokens are shown in Fig. 2. The events of a CP-net are represented by means of *transitions*, which are drawn as rectangles. The *arc inscriptions* determine which tokens are removed and added when transitions occur.

Let us consider what happens when the sender accepts a packet from the upper layers. Incoming packets from the upper layers are added to a packet buffer (Send). A token on the place Send models the buffer, and the token carries a data value that is a list of strings, where each string represents the contents of a packet. The sender can only accept (Accept) a new packet from the upper layer after an acknowledgment has been received for the previous packet. If an acknowledgment has been received, then status of the sender (NextSend) indicates the sequence number for the next packet. When a packet is accepted, a sequence number is added to the packet to form a data frame which then is ready to be sent (Ready). The status of the sender is changed to reflect that it is sending a packet that has not yet been acknowledged. It takes 5 units of time to process each incoming packet from the upper layers. A parameter in the model indicates whether the periods between packet arrivals are constant or exponentially distributed. Another parameter indicates the average amount of time that passes between the arrival of two successive packets. The rest of the Sender module works in a similar manner.

For this system, there are several performance measures of interest, including average queue length of packets waiting to be sent, average packet delay and network utilization. Packet delay is the time from which a packet is put in the Send buffer on the sender side until it is properly received by the receiver. The utilization of each direction of the bi-directional channel can be measured. Changing the values of parameters in the model will have profound effects on the performance of the model. Section 4 describes how the performance of systems can be analyzed using CP-nets.

## 3 Monitoring Model Behavior

A number of libraries and tool extensions (see [6] for references) have been developed for Design/CPN during the past decade. These libraries include support for message sequence charts (MSC), updating domain-specific graphics, and data collection. These facilities can be used both to inspect and to control a simulation of a CP-net. While there are many advantages to having this extra functionality, there are a number of disadvantages as well. The most serious problem is that it is often necessary to modify a CPN model in order to be able to use the libraries mentioned above. Modifying a model may have unexpected and undesirable effects on the behavior of the model.

Ideally in a simulation tool, there should be a clear distinction and separation between modeling the behavior of the system and monitoring the behavior of the model. With

new so-called *monitors*, it possible to inspect or control a simulation of a CP-net without having to modify the model. A monitor is activated after each step in a simulation, and if certain conditions are met, then it will observe the current state of the model and/or the event that occurred most recently, and it will take appropriate actions based on the observation just made. Each monitor contains three functions, `check`, `observe`, and `act`, that perform these services. These functions can be either predefined or user-defined. A number of standard monitors that can be used for any CP-net have been defined. It is also possible for a user to define monitors that are tailor-made for a specific CP-net.

**Simulation Breakpoint Monitor.** A simulation breakpoint monitor can be used to control a simulation. Any number of monitors can be defined for a given CP-net, and monitors can interact with each other. For example, one standard monitor can count the number of times the sender retransmits a data frame, a second monitor can reset the first monitor each time a new packet is accepted from the upper layers, and a third monitor can stop a simulation when it observes that the sender has retransmitted a data frame three times.

Monitors are easy to create. Typically, a user must select relevant parts of a CPN model by clicking on the appropriate places and transitions in the GUI of the CPN tool. Then template code can be generated for the `check` and `observe` functions. The user can then modify the template code to obtain the desired functionality. It is often unnecessary to modify the code that is generated for standard monitors. For example, a standard monitor can count how many times the sender retransmits packets in the stop-and-wait model. In this case the user must just select the TimeOut transition and then indicate that a simple counter monitor must be created. This monitor will increment a counter each time the sender retransmits a packet. An example of code for a monitor will be shown in the next section.

**Message Sequence Chart Monitor.** Monitors can also be used to create MSCs which are particularly useful when analyzing the behavior of network protocols, since they can be used to illustrate the transmission of messages. They are also very useful for debugging CPN models. Figure 3 shows an MSC that was generated during a simulation of the stop-and-wait model in Design/CPN. The MSC contains a new type of arrow which is drawn after two different events have occurred. The slope of the arrows indicate the passage of time. Arrows in traditional MSCs are always horizontal, and they are drawn when one event occurs.

When using monitors to draw MSCs, all of the functions that are used to update an MSC are gathered together, rather than being spread throughout the CP-net. With monitors, it is possible for the user to get an overview of the places and transitions that a monitor observes without having to examine all places and transitions in the CP-net. With monitors it is also very easy to inspect the state of a CPN model during a simulation, whereas in the past it was only possible to observe the events that occurred. It is not necessary to modify a CP-net to use monitors.

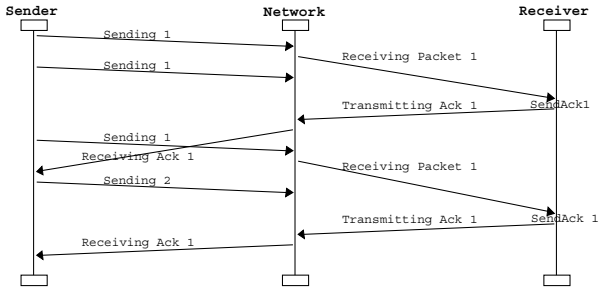


**Figure 3. Message sequence chart.**

## 4 Performance Analysis using CP-nets

This section discusses the new facilities that provide support for doing simulation-based performance analysis using CP-nets. Since a typical CPN user is not likely to be an experienced data analyst, the new facilities have been designed to assist CPN users in defining and running statistically reliable simulation experiments.

### 4.1 Data Collection

Each performance measure is calculated by a *data collection monitor*. The simulator and performance facilities for Design/CPN are implemented in Standard ML [13] (SML). Figure 4 shows the code (that has been edited for clarity) for a data collection monitor that calculates the average number of packets in the sender's packet buffer. The monitor is named `PacketQueue`.

It is not important to understand the details of Fig. 4, but the functionality of the monitor will be described in general terms. In order to calculate the average number of packets in the buffer it is sufficient to measure the number of packets in the buffer only when the number of packets changes, i.e. either when a new packet is added to the buffer or when a packet is removed from the buffer. The `Event` data type shows that this monitor only observes two kinds of events: Generate and Accept events. The monitor will be activated only when the transitions Generate (in the Upper Layers Send module, not shown) and Accept occur. For this monitor, the `check` function only needs to return `true` each time it is evaluated, i.e. it will return true when either the Generate event or the Accept event occurs. The `observe` function is then called to measure the number of packets in the buffer, and this is achieved by measuring the length of

```
structure PacketQueue  = struct                                  1
  datatype Event =                                               2
      Generate of int * {i, packets}                             3
    | Accept of int * {sn, packets, p, dframe}                   4
                                                                 5
  fun check (event, (SendMark: PacketBuffer list)) =             6
            true;                                                7
                                                                 8
  fun observe (event, (SendMark: PacketBuffer list)) =           9
      if (length SendMark)>0                                     10
          then length(hd SendMark)                               11
      else 0;                                                    12
                                                                 13
  fun act (observedval, (SendMark: PacketBuffer list)) =         14
        DataCollection.update(index, observedval);               15
                                                                 16
  fun monitor (event: Event, moduleInstance) =                   17
      let val SendMark =                                         18
                PlaceSend.getMarking(moduleInstance)             19
      in  if check(event, SendMark)                              20
            then act(observe(event, SendMark),                   21
                 (SendMark))                                     22
          else ()                                                23
      end;                                                       24
end                                                              25
```

**Figure 4. Data collection monitor.**

the one list that is found on the place Send. Most of the code in Fig. 4 is generated completely automatically. The user is only responsible for defining the bodies of the `check` and `observe` functions, represented by lines 7 and 10-12, respectively.

Standard data collection monitors that can be used to calculate a variety of different performance measures have been defined. For example, one standard data collection monitor calculates the average number of tokens on a place during a simulation, another counts the number of times a particular transition occurs. The individual data values that are observed by a data collection monitor can be saved in an *observation log file*, and they can be used to calculate statistics.

## 4.2 Analysis of One System

Performances measures that are calculated via simulation modeling are only estimates of the true performance measures. One of the dangers of using simulation for performance analysis is accepting the output statistics from a single simulation of a model as the "true answers". To compound the problem even further, analysis of output data from one simulation is sometimes done using statistical formulas that assume independence when in fact the data is dependent. New features have been developed that provide support for properly analyzing the performance of a CP-net.

**Multiple Simulations.** One of the desirable features for simulation modeling tools is a single command to make several simulation runs of a given model. A new batch script (which is just an SML function) can be used to run a given number of independent, terminating simulations in Design/CPN. Data is automatically collected and saved during each simulation. Each terminating simulation can provide one estimate for each of the performance measures that have been defined for a particular model.

**Confidence Intervals.** *Confidence intervals*[1], which indicate how precise estimates of performance measures are, are calculated when using the batch script from above. *Batch data collection monitors* are created before running a number of simulations, are updated after each simulation, and are then used to calculate confidence intervals which will be saved in a *batch performance report*.

**Simulation Output.** Simulation output is used both for analyzing the performance of the system and for presenting the results of the analysis. At the end of a single simulation, all statistics from the simulation can be saved in a *simulation performance report*. Both simulation and batch performance reports can now be saved in plain text, LaTeX and HTML formats. Additional facilities can be used to create a simple, yet organized directory system containing simulation output. Several gnuplot [7] scripts are now generated for plotting the contents of observation log files. Additional new files contain the IID[1] data that is used for calculating confidence intervals.

## 4.3 Comparing Alternative Configurations

Another batch script has been developed for running simulations for a number of different configurations of a CPN model, assuming that a new configuration can be specified by changing numerical parameters in the model. The user must specify a range of values that one or more parameters should take on, and the batch script will ensure that system parameters are changed between simulations, and that a given number of simulations are run for all given combinations of parameter values. A *configuration status file* contains the values of the parameters for each configuration, and it indicates which directory contains the output for each configuration.

One well-known technique for comparing two alternative system configurations is to calculate the so-called *paired-t confidence interval*[1] for the expected difference for a given performance measure. A paired-t confidence interval reveals whether or not the performance of two configurations are significantly different based on the available observations, and paired-t confidence intervals can easily be calculated by post-processing standard output files with an external application.

[1] For further details, see e.g. [12].

### 4.4 Variance Reduction

One of the drawbacks of simulation analysis is that it can take a long time to run a simulation. This problem is amplified if many simulations need to be run in order to achieve desired confidence intervals. *Variance-reduction techniques*[1] (VRT) can sometimes be used to reduce the number or length of simulations that need to be run. *Common random numbers*[1] (CRN) is a useful and practical variance-reduction technique when comparing alternative system configurations . The idea of CRN is to use the same random numbers for each of the configurations being studied. The effects of CRN may be improved if the simulator can be forced to use the same random numbers for the same purpose in each configuration. This process is called *synchronizing*[1] the random numbers. Not only is CRN a useful VRT, but it also implies that a fairer comparison of the configurations can be made because the experimental conditions are the same for each configuration. Support has been added for using CRN when simulating CP-nets precisely because of the appeal of this notion of fairer comparisons.

The easiest way to implement synchronization is to use different sources of random numbers for each random input process. Therefore, the random number generator that is used to generate random variates in Design/CPN has been modified, such that it can provide 10 streams of random numbers with one million independent random numbers in each stream. The random seeds for each of the streams can be reset. For example, in the stop-and-wait model a certain degree of synchronization can be achieved by using one stream for generating arrival times and another stream for generating network delays.

## 5 Conclusions and Future Work

This paper has presented an overview of improved facilities supporting simulation-based performance analysis using coloured Petri nets. With monitors it is possible to make an explicit separation between modeling the behavior of a system and observing the behavior of a system. As a result, cleaner, more understandable CPN models can be created, and the risk of introducing undesirable behavior into a model is reduced. New facilities have been created for running multiple simulations, generating statistically reliable simulation output, comparing alternative system configurations, and reducing variance when comparing configurations. Most of the facilities presented here have been implemented, however, some have been implemented for Design/CPN and others for CPN Tools [5], which will eventually replace Design/CPN. Therefore, not all of them work together. Since CPN Tools will be the successor to Design/CPN, a current project is working on updating and porting the facilities from Design/CPN to CPN Tools, and the performance-related facilities will be incorporated into CPN Tools as part of this project.

There are certain disadvantages associated with using simulation based performance analysis: no definitive answers can be provided, and it may take a long time to run enough simulations in order to calculate sufficiently accurate performance measures. However, it is the best alternative for analyzing the behavior of industrial-sized models.

## References


[1] E. Brinksma, H. Hermanns, and J.-P. Katoen, editors. *Lectures on Formal Methods and Performance Analysis*, volume 2090 of *Lecture Notes in Computer Science*. Springer, 2001.

[2] L. Cherkasova, V. Kotov, and T. Rokicki. On scalable net modeling of OLTP. In *Proceedings of 5th International Workshop on Petri Nets and Performance Models*, pages 270–279. IEEE Computer Society Press, 1993.

[3] S. Christensen et al. Design/CPN - a computer tool for coloured Petri nets. In E. Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems - TACAS'97*, volume 1217 of *Lecture Notes in Computer Science*, pages 209–223. Springer-Verlag, 1997.

[4] H. Clausen and P. R. Jensen. Analysis of Usage Parameter Control algorithms for ATM networks. In S. Tohmé and A. Casada, editors, *Broadband Communications II (C-24)*, pages 297–310. Elsevier Science Publishers, 1994.

[5] CPN Tools. Online: http://wiki.daimi.au.dk/cpntools/.

[6] Design/CPN. Online: http://www.daimi.au.dk/designCPN/.

[7] gnuplot. Online: http://www.gnuplot.info/.

[8] S. Gordon and J. Billington. Analysing the WAP class 2 wireless transaction protocol using coloured Petri nets. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 207–226, 2000.

[9] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts*. Monographs in Theoretical Computer Science. Springer-Verlag, 1997. 2nd corrected printing.

[10] W. D. Kelton, R. P. Sadowski, and D. A. Sadowski. *Simulation with Arena*. McGraw-Hill, 2nd. edition, 2002.

[11] L. M. Kristensen, S. Christensen, and K. Jensen. The practitioner's guide to coloured Petri nets. *International Journal on Software Tools for Technology Transfer*, 2:98–132, 1998.

[12] A. M. Law and W. D. Kelton. *Simulation Modeling & Analysis*. McGraw-Hill, 3rd edition, 2000.

[13] L. C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 2nd edition, 1996.

[14] Petri Nets Tool Database.
Online: http://www.daimi.au.dk/PetriNets/tools/db.html.

[15] M. Villapol and J. Billington. Modelling and the initial analysis of the Resource Reservation Protocol using coloured Petri nets. In K. Jensen, editor, *Proceedings of the Workshop on Practical Use of High-Level Petri Nets*, DAIMI PB-547, 2000.

[16] L. Wells et al. Simulation based performance analysis of web servers. In R. German and B. Haverkort, editors, *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, pages 59–68. IEEE, 2001.