Ain Shams University
Faculty of Engineering
Computer and Systems Department

REAL TIME COMMUNICATION

A Thesis
Submitted in Partial Fulfillment of the
Requirements of the Degree of Doctor of
Philosophy in Electrical Engineering
(Computer and Systems Engineering)

Submitted by
Khalid Mohamed Nabil Mahmoud
Master Degree in Electrical Engineering
(Computer and Systems Engineering)
Ain Shams University, 1996

Supervised By
Prof. Dr. Yasser H. Dakroury

Cairo, Egypt
May - 2006

APPROVAL SHEET

Name: Khalid Mohamed Nabil Mahmoud
Thesis: Real Time Communication
Degree: Doctor of philosophy in Electrical Engineering (Computer and Systems Engineering)

Examiners Committee

Name, Title, and Affiliation **Signature**

1. Prof. Dr. Mohamed Gamal El Deen Darwish
Professor, Faculty of Computers,
Cairo University

2. Prof. Dr. Mohamed Adeeb Riad Ghonaimy
Professor, Ain Shams University

4. Prof. Dr. Yasser Hisham Dakroury
Professor, Ain Shams University

Date : 20/04/2006

Thesis Title: Real Time Communication

Name: Khalid Mohamed Nabil Mahmoud

Degree: Doctor of philosophy in Electrical Engineering (Computer and Systems Engineering)

Abstract

Protocols responsible for Quality of Service (QoS) routing are characterized by limited scalability, considerable complexity and traffic overhead. QoS routing implies resource reservation to support the application requirements. The implementation of QoS routing and resource reservation require signaling mechanisms. Performing QoS routing and resource reservation at different network layers will cause redundant signaling. In this work, we propose a simple extension to the Resource Reservation Protocol (RSVP) signaling mechanism that supports the simultaneous implementation of QoS probing routing and resource reservation. Supporting a resource reservation protocol as RSVP with QoS routing facility, to reserve resources at routing time, solves the problem of uncertainty of the network node states. The extended RSVP uses only the IP protocol and eliminates the need of additional signaling mechanism in the network layer. The original RSVP is not capable to use multi paths provided by multi-path routing at network layer. In our proposed approach, multi paths are used to route the traffic if QoS requirements cannot be satisfied using a single path.

This thesis provides two methodologies: The first is concerning finding single path. The second is concerning finding multi path. In both methodologies, the following work is done:

The proposed approach aspects are first formalized. The functions of the extended protocol within a sender, a router, and a receiver are determined and formalized using Colored Petri net (CPN) notation. The sender, router, and receiver are modeled using CPN tool. In this model, the dynamic change of the network resources and delays are simulated by probability values.

Verification of the extended signaling mechanism is accomplished by constructing a simple verification network and analyzes the state space of this mechanism. Proper sequence, termination, and loop free are verified. Verification of selection process of a single path or multi paths is also examined. Performance evaluation is presented to show the probability of success gained from the extended protocol. A simulator of CPN modeled sender, receiver and routers has been constructed. This simulator is used to measure the performance of the extended protocol at different network loads. We proved that the total success probability is equal to the sum of the success probability of individual probes. Even when a success path is not found, we still can use a collected QoS of multi paths. We showed that a limited increase of used probes significantly increases the total probability of success.

Statement

This dissertation is submitted to Ain Shams University the Degree of Doctor of Philosophy in Electrical Engineering (Computer and System Engineering).

The work included in this thesis was out by the author at the Computer and System Engineering Department, Ain Shams University.

No part of this thesis has been submitted for a degree or qualification at other university or institution.

Date: / / 2006
Signature:
Name: Khalid Mohamed Nabil Mahmoud

Acknowledgements

First of all I thank God for all his generosity and help.

I would like to express my deep gratitude to my supervisor Prof. Yasser Dakroury for his valuable advises, and his great guidance for me throughout this work. He actually gave me the chance to express my capabilities. I am really grateful for his patient and gentility.

I would like also to thank Prof. Mohammad Adeeb Ghonaimy for his kind deep revise and remarkable notes which help my work to be appear in best shape.

I would like to thank my teachers, Staff of Computer and Systems Engineering Department for all what they taught me and also all my fellows for their support and encouragement.

I would like also to thank my family for their deep prays and support for me during the thesis development.

Finally I would like to thank my mother and beloved wife for her patience and great support and encouragement during the most important stages of the thesis.

Appendix - Colored Petri net

Colored Petri nets CPN [117][122] represent a graphical structure tool with associated computer language statements. It provides a framework for the design, analysis, validation, and verification of systems. CP-nets have a wide range of application areas for e.g., in the areas of communication protocols, operating systems, hardware designs, embedded systems, software system designs, and business process re-engineering.

The principal components of a CP net are:

• **Data**: CP nets make use of data types, data objects, and variables that hold data values. CPN data is defined using a computer language called "CPN ML".

• **Places**: Locations for holding data.

• **Transitions**: Activities that transform data.

• **Arcs**: Connect places with transitions, to specify data flow paths.

• **Input Arc Inscriptions**: Specify data that must exist for an activity to occur.

• **Guards**: Define conditions that must be true for an activity to occur.

• **Output Arc Inscriptions**: Specify data that will be produced if an activity occurs.

1- CPN Data

CP nets use data objects, data types, and variables. CPN data objects called "Tokens" while the data types are called "color". All CPN data types and variables must be declared.

Color declarations

CP net tokens can be any of all the data types generally available in a computer language: integer, real, string, Boolean, list, record, and so on. The

CPN defines the set of values or elements which a color could be assigned using the word "with". For example, in table 1 chapter 5.

color NetNodes=with E|S|D|R1|R2|R3;

the NetNodes color could be one of E,S,D,R1,R2, or R3.

Color declarations may use expressions of previously declared colors. It could be same color as previously declared one .for example

color PreN= NetNodes;

color NextN= NetNodes;

or a subset of previously defined set using the word "subset", for example

color DestNodes=subset NetNodes with[D];

DestNodes is a sub set of NetNodes set with element D.

CPN also defines Tuples and Union colors. The Tuple is ANDing of two previously defined colors using * operation. For example

color PRoute = product PreN * NextN * DestNodes;

while Union is ORing of two previously defined colors using + operation . For example

color UpMsg = union patherror:patherrorP + resvtear:ResvTearP + resv:resvP;

the UpMsg could be one of several colors representing up stream messages.

Multi sets of Tokens

In dealing with CP nets, it is often necessary to manipulate and refer to collections of tokens. A collection of zero or more tokens is called a *multi set*. A multi set may contain multiple tokens that have the same value. All tokens in a multi set must be of the same color set. Any subset or union of multi sets is again a multi set.

Specifying Variable Token Values

Sometimes it does not matter exactly what values enabling tokens have. Input arc inscriptions use CPN variables rather than constants to carry their color's tokens to or from a transition when it is enabled (fired). These variables are

called *CPN variables*. Like color sets, CPN variables must be explicitly declared. Every CPN variable is of a particular color set, and can take on only values of that type. When a CPN variable has taken on the value of some token, it is said to be bound to that value. When a CPN variable is not bound to any value, is said to be *unbound*. For example

var p,P:Prob;

Define p and P two variables of Prob color.

2- CPN Components

Places

Tokens would be of little use if they could not be stored and accessed as needed. CP nets keep tokens in locations called places. A *place* is a location that can contain zero or more tokens of some particular color. The token set in a place is called the *marking* of the place. A place always has a marking: if it contains no tokens, its marking is empty. A place's color is one of the colors defined for the net. All tokens in a place must be of that color. A place may optionally have a name. Such a name is useful for indicating what the place means as a part of a model, for identifying the place when humans confer about the net, and for labeling computer- generated information about activity in the place during simulation runs.

A place is graphically represented as an ellipse. Its name and/or color may optionally be displayed next to or inside the place. They are graphically represented as labels that are regions of the place.

Transitions

A *transition* is an activity whose occurrence can change the number and/or value of tokens in one or more places. A transition may optionally have a name, kept in a name region, which serves the same purposes as a place name.

A transition is graphically represented as a rectangle. If it has an associated name, it may optionally be displayed next to or inside the transition.

Guards

A *guard* is a Boolean expression associated with a transition. Guards are customarily written inside square brackets, to help distinguish them from other regions. A transition's guard must evaluate to Boolean **true** in order for the transition to occur. A guard need not be given explicitly. The default guard is Boolean **true**. When a guard is given as or defaults to **true**, the guard never restricts the transition from occurring.

Arcs

An *arc* is a connection between a place and a transition. Every arc connects one place with one transition. Every arc has a direction, either from a place to a transition or from a transition to a place. An arc is represented graphically as an arrow with a head that indicates its direction. An arc that runs from a place to a transition is called an *input arc*, and the place it connects to be called an *input place*. An arc that runs from a transition to a place is called an *output arc*, and the place to which it connects is called an *output place*. It is possible for a place to be both an input place and an output place.

Arc Inscriptions

An *arc inscription* is a multi set expression of tokens or their variables associated with an arc. It is kept in a region of the arc called an *arc inscription region*. An arc inscription on an input arc is called an *input arc inscription*; an arc inscription on an output arc is called an *output arc inscription*. The tokens in the multi set specified by an input arc inscription must be present in the input place in order for the transition to occur (there may be other tokens there also). These tokens will be removed from the input place if the transition does

occur. The multi set specified by an output arc inscription will be put into the output place if the transition occurs.

An arc inscription need not be given explicitly. The default arc inscription is empty, the multi set of no tokens. When an input arc inscription is given as or defaults to empty, no tokens need to be in the input place in order for the transition to occur. When an output arc inscription is given as or defaults to empty, no tokens will be put into the output place if the transition does occur.

The statement "if" could be used in the arc inscription chose different multi sets or empty set according to a condition. For example, in CPN RSVP Router graph, the inscription of the output arc from "ResvError" transition to place "R_C_Link" is one "resverror" token if the "status" token has value equal to "TempResv" otherwise no token output.

3- CP Net Execution

A CP net is more than a description of system structure: CP nets are intended to be executed. Such execution can provide information about system dynamics that could never be derived by looking at a static model and considering its implications. There is just too much information involved in the functioning of a complex system for the unaided human mind to cope with.

Transition Occurrence

A transition can *occur* whenever certain conditions are met. When the conditions are met, the transition is said to be *enabled*. The fact that a transition is enabled does not mean that it will actually occur: some other enabled transition might occur first, and change the state of the net so that the first transition is no longer enabled.

Three factors work together to determine whether a transition is enabled:

1. Input Place Multi set: The multi set of tokens in each input place of the transition. The multi set of tokens present in an input place determines what tokens are available when the simulator is determining if a transition is enabled.

2. Input Arc Inscriptions: The input arc inscription specifies a multi set on each input arc connected to the transition. The inscription may be empty which the default value is.

3. The transition's Guards: The boolean expression that is associated with a transition. This expression must evaluate to **true** in order for the transition to be enabled. The default guard is **true**.

These three factors work together to determine whether a transition can occur. They work together so closely that none of them can be fully understood without understanding the other two.

When a Transition Occurs

The previous sections describe what is necessary in order for a transition to be enabled, that is, in order for it to be able to occur. When the simulator affects the occurrence of a transition, it is said to *fire* the transition. Not every enabled transition actually occurs, some other enabled transition might occur first, and change the net so that the first transition is no longer enabled.

Now we will examine what happens if the transition actually occurs.

1. Rebind any CPN variables as indicated by the enabling binding.
2. Evaluate each input arc inscription.
3. Remove the resulting multi set from the input place.
4. Evaluate each output arc inscription.
5. Put the resulting multi set into the output place.

Binding an input Arc Inscription Variable

Initially the CPN variable has no particular value: it is said to be *unbound*. The arc inscription that uses it does not evaluate to a multi set. The simulator's task is to evaluate the arc inscription; it chooses a value from among the possible legal values in the arc input place. When this has occurred, the variable is said to be *bound* to that value.

Evaluate Each Output Arc Inscription

Output arc inscription are evaluated by bounding the CPN variables of the inscription.

Constraining Token Values with Guards

In order to model real systems, we need a way to require token values to meet any criterion we can define. Such a requirement is called a *constraint*. In order to constrain token values, CP nets use arc inscriptions in conjunction with guards. The method is to bind CPN variables in arc inscriptions, and perform boolean tests on those values in a guard.

4- CPN Tool

CPN Tool is a computer tool that supports the construction, simulation, functional and performance analysis of CPN models. The tool was developed by the CPN group of the University of Aarhus, Denmark and Metasoft. CPN Tool is an enhanced user interface of the old version Design/CPN [123]. CPN Tool includes two main components: the *Graphical User Interface (GUI)* and *CPN ML* [118] the user creates and manipulates CPN models through the GUI. CPN ML implements the programming language used for the declaration of variables, declarations of types and net inscriptions, such as arc expressions and guards. CPN ML is also used to analyse the CPN model as will be described. CPN ML is a dialect of the *Standard ML* language. The syntax and

semantics of CPN ML are supported by the tool. CPN Tool can run over Windows platform as we use it in this thesis.

Hierarchical CPNs

Hierarchical CPNs allow models to be built in a modular and structured way. Hierarchical levels show different levels of detail of the model. For example, the Sender, Receiver, and Routers in Network simulator Figure 5-7 are an abstraction of detailed model in Figures 5-3, 5-4, 5-5 In this way, hierarchical CPN allows the construction of large and complex models in a more compressive manner.

A CPN model can be structured into a number of pages. Hierarchical CPNs provide two mechanisms to interconnect these pages: *substitution transitions* and *fusion places*. *Substitution transitions* allow models to be built in a top-down manner. If a transition is a substitution transition, it has a sub-page related to it, which includes a more detailed description of the model. *Fusion places* include a set of places, which are functionally identical, so they have the same marking. A set of fusion places is a *fusion set*. Members of a fusion set can belong to a single page or be distributed across different pages. *Substitution transitions* and *fusion places* are used to constructs the Network Simulations Figure 5-7 and 6-5.

5 - Analysis of CPNs

The CPN model can be simulated. Simulations allow the user to understand and to debug the model. Several analysis methods have been created to obtain the properties of the model.

In this thesis, **the *State Space*** method is used. The state space includes all possible markings that can be reached from the initial marking. It is represented by a directed graph where the nodes represent the markings and the arcs the occurring binding elements. The state space is also called an *occurrence graph* or *reach ability Graph*. States spaces can be automatically

constructed, which provides computer-aided analysis and verification of the behavior of the modeled system, and can be used for debugging and testing the system. In some cases, the model of the system can generate a very large or even infinite number of markings. In such cases, the state space cannot be generated, however some properties can be proved or disproved based on partial state spaces including finite sub-graphs of the full state space. The state space tool is fully integrated in CPN Tool.
Figure 5-6 shows the state space for the simplified model of the proposed protocol.

A *strongly connected component* *(SCC)* [120] of the state space is a maximal sub-graph, whose nodes are mutually reachable from each other. A SCC graph has a node for each SCC and arcs that connect each SCC node with other SCCs. A SCC without incoming arcs is called the *initial SCC*, and a SCC without outgoing arcs is called a *terminal SCC*. Each node in the state space belongs only to one SCC, so the SCC graph will be smaller than or equal to the corresponding state space.

Behavioral Properties of CPNs

The properties that describe the expected behavior of the model can be defined. In this section, the main behavior or dynamic properties of CPNs are described in an informal way. More details about these properties and their formal definitions can be found in [117] [122].

Reach ability:

By convention, Mn denotes the marking of node number n. Mn is *reachable* from M1 if there is an occurrence sequence from marking M1 to Mn. For example, in Figure 6 chapter 5, M30 and M31 are reachable from M13 and not reachable from each other.