Modeling and Analysis of Software Rejuvenation in Cable Modem Termination System

Yun Liu †, Yue Ma ‡, James J. Han †, Haim Levendel ‡ and Kishor S. Trivedi †
† CACC, Department of ECE, Duke University, Durham, NC 27708, USA
liu, kst@ee.duke.edu
‡ Global Software Group, Motorola Inc., Elk Grove Village, IL 60007, USA
a14438, cjh048, ail003@email.mot.com

Abstract—In order to reduce system outages and the associated downtime cost caused by the “software aging” phenomenon, we propose to use software rejuvenation as a proactive system maintenance technique deployed in a CMTS (Cable Modem Termination System) cluster system. Different rejuvenation policies are studied from the perspective of implementation and availability. To evaluate these policies, stochastic reward net models are developed and solved by SPNP (Stochastic Petri Net Package). Numerical results show that significant improvement in capacity-oriented availability and decrease in downtime cost can be achieved. The optimization of the rejuvenation interval in the time-based approach and the effect of the prediction coverage in the measurement-based approach are also studied in this paper.

I. INTRODUCTION

Cable modem has emerged as one of the most popular broadband access technologies due to the widespread two-way Hybrid Fiber Coaxial (HFC) cable networks and the industry standard: Data Over Cable Service Interface Specifications (DOCSIS) [4]. It is expected to be the predominant high speed Internet access technique for delivering data, voice, and video services to the homes in the next ten years.

A cable modem system consists of three major components: HFC cable network, cable modem (CM) at each customer location on one end of the cable plant, and cable modem termination system (CMTS) at the headend on the other end of the cable plant (Figure 1). One of the primary functionalities of CMTS is to provide an interface between the cable network and the Internet for both upstream and downstream traffic. Furthermore, CMTS is also responsible for managing system operations such as billing, subscriber authorization, and quality of service (QoS) control. Therefore, CMTS plays a central role in the cable modem network and involves very complex hardware and software implementations. The high availability of CMTS is crucial for cable operators to provide carrier-class services to all subscribers.

In traditional approaches, high availability is achieved by introducing hardware redundancy and the corresponding resilient software features. As shown in Figure 1, a high availability CMTS is usually built based on a cluster architecture. The cluster comprises N primary CMTS (PCMTS) nodes, one secondary CMTS (SCMTS) node, and an HFC interface switch (HIS). Each PCMTS node connects to a certain group of HFC fibers providing access service for all the customers residing in the covered area of those fibers. Different PCMTS nodes in the cluster connect to different HFC fibers and thus serve different customers. The redundant SCMTS node normally stays in a warm or hot standby state. When any hardware/software failure of a PCMTS node is detected, a two-step switchover action is invoked. First, the HIS switches the fiber connection from the failed PCMTS node to the SCMTS node, which continues all the ongoing operations of the failed node and acts as the original PCMTS. Second, the failed PCMTS node is repaired offline and a reverse switchover is performed by the repaired PCMTS node to take back its operations from the SCMTS node. The switchover is made possible by the SCMTS node replicating all the necessary run-time states for every PCMTS node in the cluster. This requires a database distribution and synchronization process that resides on both PCMTS and SCMTS nodes.

Although the fault-tolerant design discussed above may significantly increase system availability and reduce downtime cost, there exist some inherent limitations that leave room for further improvement. First, the success of the traditional high availability techniques relies on a quick and almost error-free failure detection and recovery mechanism. As this is hard to realize in practice, the system availability is not as high as it is claimed in the ideal situation. Second, traditional fault tolerance techniques are reactive in nature. As the corrective action is taken after the failure has occurred, it usually involves considerable system maintenance cost and potential financial losses. However, if some proactive fault management can take appropriate actions before the system experiences failures, both system unavailability and downtime cost can be reduced in a cost-effective way. Specifically, this proactive technique is called
Software rejuvenation, which is proposed to counteract the software aging phenomenon [1] [12] [14] [18].

Software rejuvenation in cluster systems has recently been studied in [19]. The main contribution of their work is to show that using software rejuvenation can significantly improve cluster system availability. Our work is quite different since the CMTS cluster system in our study has completely different conceptual and practical meaning. In [19], all the $N$ nodes in the cluster execute a cluster server software. The cluster is defined as available when there are less than $a$ ($a \leq n$) individual node failures. However, in a CMTS cluster, each node is connected with a specific group of customers. The failure of a node can only cause service unavailability for its own customers but has no impact on the customers served by other nodes. As a result, the definitions of availability and cost measures are different (Section IV-D). In addition, we also consider node failures caused by hardware faults and Heisenbugs (Section II-A) rather than only by aging-related software faults. The system recovery behavior in the presence of node failures is much more complex due to the switchover action required for service continuity in CMTS.

Note that software rejuvenation technology has been incorporated into the IBM Director for xSeries servers because of the system availability improvement and cost reduction [3]. This is the first commercial application of software rejuvenation. Nevertheless, no CMTS cluster systems have been observed to introduce this preventive maintenance technique as far as we know. As discussed in Section II and Section III, the phenomenon of software aging generally exist in almost all computer systems, especially in systems with huge and complex software. In addition, the high availability architecture of CMTS cluster provides a convenient framework that can be easily modified to deploy different software rejuvenation schemes. The authors of this paper hope that the encouraging numerical results of the analytic models may expedite the use of software rejuvenation technology in the future high availability CMTS systems as well as other non-CMTS cluster systems.

This paper studies the availability and downtime cost of a CMTS cluster with and without software rejuvenation. Moreover, different rejuvenation policies are discussed and their performances are compared in terms of availability improvement and downtime cost reduction. Continuous Time Markov chain (CTMC) models are used to analyze various policies. The task of construction of CTMC models is simplified by using a higher level paradigm known as stochastic reward net (SRN). Stochastic reward net models are solved by using SPNP (Stochastic Petri Net Package) [2]. The benefit of introducing software rejuvenation is quantified by the numerical results. Furthermore, the optimization of the rejuvenation interval in time-based rejuvenation scheme and the effect of the prediction coverage in measurement-based rejuvenation scheme are also studied in this paper.

The rest of the paper is organized as follows. In Section II, classification of software faults is given and the concept of software rejuvenation is reviewed. In Section III, the outline for implementing software rejuvenation in a CMTS cluster environment is proposed. In Section IV, SRN models are developed for different CMTS configurations. In addition, availability and cost measures are provided for each analytic model. Numerical results are discussed in Section V. Finally, conclusions are drawn in Section VI.

II. SOFTWARE FAULTS AND SOFTWARE REJUVENATION

A. Classification of Software Faults

Due to the explosive demand for reliable computing systems, hardware fault-tolerant techniques have been heavily investigated and the corresponding hardware failure rates have been dramatically reduced in recent years. The hardware components in CMTS can have a mean time to failure (MTTF) as large as hundreds of thousands of hours (i.e., tens of years). On the other hand, as the growth in software complexity and...
reusing. As a result, it is very common for current commercial software to have an MTTF ranging from hundreds to thousands of hours, especially under the pressure of rapid product release. Therefore, it has been well established that system failures are much more frequently caused by software faults rather than hardware faults [11][16].

Jim Gray has suggested classifying software faults into two categories, Bohrbugs and Heisenbugs [10]. Bohrbugs are easy to reproduce and detect. An operation containing Bohrbugs will always cause error when it is retried. Bohrbugs should ideally have been identified and removed during the testing and debugging phase. Otherwise, the only way to avoid errors caused by Bohrbugs during operation is to provide design diversity where the functionality is realized through different design/implementations. Unlike Bohrbugs, Heisenbugs are difficult to reproduce and detect since they are only revealed under rare system conditions. Errors caused by Heisenbugs usually do not reoccur on retry since the system state is slightly changed. Therefore, Heisenbugs are said to be transient and errors caused by Heisenbugs can be corrected by retrying the same operation or restarting the system.

In the study of the “software aging” phenomenon [12][15][18][21], the third type of software faults, aging-related bugs, has been recently proposed to extend the original classification (Figure 2). Software aging refers to the fact that error conditions can accrue with time and/or load. Typical causes of software aging include memory leaking, unreleased filelocks, data corruption, file descriptor leaking, and storage space fragmentation. Gradually accumulated potential fault conditions may lead to performance degradation, transient failures, or both. Failures may be crash or hang type, both may lead to the impairment of availability and increased maintenance cost. Although software failure due to resource exhaustion can be recovered through restarting, reactive action might involve great financial loss. Therefore, a proactive software maintenance was proposed to deal with aging-related bugs, which is known as software rejuvenation [9][12].

B. Software Rejuvenation

Software rejuvenation is the technique aimed at reducing system outages caused by the aging-related bugs. This technique generally involves stopping a running program occasionally, cleaning system internal states, and restarting the program. According to the control mechanism, software rejuvenation can be categorized into two approaches, open-loop control and closed-loop control [15]. Open-loop approach is characterized by no feedback information from the system after the integration of software rejuvenation functionality. Time-based rejuvenation and its variants fall into this category. On the other hand, in closed-loop approach, rejuvenation trigger is dependent on some form of feedback from the system. The rejuvenation decision is made based on current system state and/or previous system behavior, which include workload, resource usage [7], and failure logs. Measurement-based rejuvenation belongs to this category [20].

1) Time-based Rejuvenation: This rejuvenation policy is characterized by the fact that the software is rejuvenated every time a predefined time constant $\delta$ has elapsed.

The primary challenge in time-based rejuvenation is to determine the value of $\delta$ so that the optimization is achieved in terms of minimum downtime or cost. As the downtime caused by rejuvenation is scheduled, the associated cost is usually much less than that of unplanned system outages. However, rejuvenation might incur increase in both node downtime and cost when performed excessively. Therefore, the rejuvenation interval is a critical parameter in time-based rejuvenation.

A variant of pure time-based scheme is to consider system load. Under certain circumstances, the objective may not be simply minimizing system unavailability or cost. For example, in a transaction-based system, we may be more interested in reducing the number of rejected transactions when the system experiences outages. In this case, both the elapsed time $\delta$ and the traffic load should be considered jointly [9].

2) Measurement-based Rejuvenation: This rejuvenation policy is also referred to as prediction-based rejuvenation since rejuvenation decision is made based on monitored system parameters. Usually, these selected parameters describe the usage of certain system resources such as free memory space. From the collected parameters, software aging and other anomalies can be detected through appropriate statistical techniques [3]. Furthermore, the system failure time due to resource exhaustion can be estimated by the smoothing and local regression trend detection techniques [7].

When this scheme is applied to manage a group of nodes in a cluster, the Simple Network Management Protocol (SNMP) could be used to monitor system parameters in a client-server manner. The manager, the agent, and the Management Information Base (MIB) are the three basic components to construct an SNMP-based rejuvenation management architecture. The manager resides in the central monitor node and periodically sends requests to the agents running on the monitored nodes for their instantaneous state parameters. From the collected data, statistical techniques can be applied to detect the software aging, predict the failure time for each monitored node, and use this information to rejuvenate system (or its resources) before a software failure occurs [7].
A crucial parameter of this policy is the probability of successful aging failure prediction. As the prediction of resource exhaustion may not be accurate from previous system parameter samples, there exists a possibility that the manager does not indicate an agent to perform the rejuvenation before the occurrence of a failure due to resource exhaustion. In this case, faults will be escalated to a higher level and the resulting node failure will be detected and recovered by the existing fault-tolerant mechanism in the system.

III. AN OUTLINE FOR IMPLEMENTING SOFTWARE REJUVENATION IN CMCT

A. Architecture and Functionality Flowchart

The proposed software rejuvenation technology naturally fits in the CMCT cluster with N+1 redundancy. As mentioned before, the standby redundancy requires the fault detection and cluster management software, which can monitor the state of each PCMTS/SCMTS node and control the switchover process when failure occurs. The design of software rejuvenation can exploit the same framework.

The software for performing rejuvenation task consists of a software rejuvenation manager (SRM) and a group of software rejuvenation agents (SRA). SRM resides on the SCMTS node and SRAs reside on each PCMTS node as shown in Figure 3. The functionality of SRM and SRA varies with different rejuvenation policies such as time-based or measurement-based schemes.

In general, an SRA is designed to respond to requests from SRM for the local resource monitoring task (Figure 4). The SRM is responsible for making the decision as to which node needs rejuvenation. For the SCMTS node, it can be rejuvenated whenever suitable, that is, when it is not involved in any switchover or repair procedure. A PCMTS node can only be rejuvenated when both the SCMTS and the HIS are available. The rejuvenation action consists of switching over to SCMTS, rejuvenating the chosen PCMTS, and switching back to the original PCMTS upon completion of its rejuvenation. The rejuvenation of a CMCT node may have different granularity, either restarting the whole operating system or restarting a specific application program.

B. Design for Time-based Policy

In this policy, the rejuvenation is triggered after a fixed time has elapsed. If the timer is maintained by SRM, no SRA is needed to implement rejuvenation. Otherwise, SRM obtains the timing information from each node reported by SRA. When a specified time interval has elapsed, a rejuvenation trigger is generated by SRM and the timer is reset to zero. The SRM needs to maintain a rejuvenation queue as multiple rejuvenation requests may be pending in the system.

The rejuvenation interval can be determined either from an analytic model [6], simulation model, or previous failure data from the field [5]. SRM can also maintain a calendar in which the system administrator can specify the days when rejuvenation is allowed and when it is forbidden [3]. This can provide more flexibility in resolving potential conflicts under system update or reconfiguration.

C. Design for Measurement-based Policy

The designs of SRM and SRAs are more complex for a measurement-based rejuvenation policy. In this scheme, SRA is responsible for monitoring the system resource parameters periodically and providing the data to SRM. In order to minimize the computation burden on PCMTS nodes, the prediction algorithm of resource exhaustion time should be implemented on the SCMTS node.

According to the study in [7], we can choose to monitor the following operating system resources realMemoryFree, usedSwapSpace, fileTableSize, and procsTotal. Furthermore, other parameters related to the file system, the network resource, and I/O devices can also be included.

Seasonal Kendall test was proposed in [7] to validate the existence of resource utilization trend in a certain time period. As it is expected to have some degree of periodicity in the duration of a day, we can set the cycle of 24 hours to perform the test. Let \( x_i \) denote the resource utilization in the \( i \)th sample and there are totally \( n \) samples in every cycle. From the Mann-Kendall statistic \( S = \sum_{k=1}^{n-1} \sum_{l=k+1}^{n} sgn(x_l - x_k) \), we can compute the significance of trend over each cycle. When the trend is indicated as positive or negative by \( S \), Sen’s slope estimate can be used to obtain the rate of resource exhaustion in each cycle and consequently the expected time to resource exhaustion. If rejuvenation is carried out before the predicted time to resource...
exhaustion, the likelihood of software crash or hang will be re-
duced.

IV. MODELING CMTS WITH SOFTWARE REJUVENATION

In this section, we construct SRN models for different sys-
tem configurations and also provide the availability and cost
measures for each case in terms of reward rate specifi-
cations. The assumptions for the models under study are given as fol-

1) Both hardware and software failures are considered in our
models. Software failure can be caused by Heisenbugs or
aging-related bugs. We assume that there are no remain-
ing Bohrbugs in the operational software.

2) Hardware failures can always be detected and then re-
paired. Software failures are transient in nature and
thus can be recovered by rebooting the whole node. No
system-wide software failures are considered, which can
bring the whole cluster down.

3) Most software failures (100c1 percent) manifest them-
selves as node level crashes or not being able to respond
to heartbeat messages and thus can be automatically de-
tected by the embedded system software in a short pe-
riod of time. However, there always exist a small portion
(100 (1 c1) percent) of software failures which can not
be revealed in an automatic manner. They usually require
a substantially longer time to detect with human inter-
vention such as customer complaints or scheduled system
maintenance. We call the first case automatic detection
and the second case manual detection.

4) We assume that a PCMTS node does not fail during
the switchover/giveback and rejuvenation actions. When the
SCMTS node participates a repair/reboot procedure, we
assume that it does not fail until the original PCMTS node
has taken all its operations back from the SCMTS node.
These assumptions are reasonable as the duration of these
actions is several orders of magnitude lower than a node’s
MTTF.

5) HIS is assumed to be a fault free device. In other words,
its failure will not have any impact on the network ele-
ments not involved in the switchover procedure and hence
the active traffic will not be affected.

6) The distribution of time between hardware failures
and software failures caused by Heisenbugs are as-
sumed to be exponential. Time between software fail-
ures caused by aging-related faults is assumed to be
hypo-exponentially distributed as is done by several
other authors [8] [12]. Failure detection time and
node switchover/reboot/rejuvenation/giveback time are
assumed to be exponentially distributed.

A. SRN Model of a basic System without Rejuvenation

Figure 5 shows the SRN model of the basic system that has
not employed software rejuvenation technique. Initially, N to-
kens are in place P_{PCMTS} in Figure 5(a), which represents
the situation that all the N PCMTS nodes are in a robust and
healthy state when the system is started. In this state, it is im-
possible for the system to encounter aging-related faults. How-
ever, failure can be caused by Heisenbugs or hardware faults,
represented by the firing of transitions T_{p,fail} and T_{p,hw}. As
time progresses, transition T_{p,aged} fires and removes one to-
ken from place P_{PCMTS} and deposits one token in place P_{aged}.
This indicates that a PCMTS node eventually enters the “soft-
ware aging” state and the node may fail due to resource ex-
haustion. Note that the PCMTS node is still operational in this
state but its performance might be degraded. Firing of transi-
itions T_{p,fail} and T_{p,hw2} represent the occurrence of software
failures (caused by Heisenbugs and aging-related faults) and
hardware failures, respectively. Upon the firing of transition
T_{p,fail}, with probability c1 the software failure is automati-
cally detected, which is represented by the path composed of
transition t3, place P_{o}, and transition T_{p,o}. Or, with proba-
bility 1 c1, the software failure can only be detected manu-
ally, which is represented by the path composed of transition
t4, place P_{m}, and transition T_{p,m}. The number of tokens in
place P_{detected} represents the number of failed PCMTS nodes
due to software failure. The SCMTS node has a similar failure
behavior as shown in Figure 5(b).

Firing of transition T_{swt} in Figure 5(a) represents the
switchover action where the SCMTS node takes over all the
running operations of a failed PCMTS node. Transition T_{swt}
can only fire when the SCMTS is available, which is guaranteed
by its enabling function. As there is only one SCMTS node in
the cluster, at most one PCMTS node can be in the repair/reboot
process at any time either due to hardware or software failure.
Firing of transition T_{p,repair} represents the reboot of the failed
PCMTS node. After the failed PCMTS node is restarted and
rejoins the cluster, the SCMTS node puts back all the opera-
tions it obtained from the failed PCMTS node and goes back to
the warm standby state. This return action is represented by the
firing of transition T_{swtbk}. The recovery mechanism is simpler
for SCMTS as it does not require the switchover and giveback
actions.

The repair process of a hardware failure differs from the re-
covery of a software failure in two aspects. First, the node needs
to be repaired (transitions T_{p,repair} and T_{s,repair}) rather than
rebooted after a failure. The repair may take as long as several
hours rather than several minutes for a reboot. Second, the au-
omatic detection (transitions T_{p,a,hw} and T_{s,a,hw}) of hardware
failures is assumed to be always successful.

Since we assume that the SCMTS node does not fail during
the whole recovery procedure, transitions T_{s,hw}, T_{s,hw2},
T_{s,fail}, and T_{s,Hei} are disabled if there is a token in place
P_{swted}, P_{swted,hw}, P_{rebooted}, or P_{repaired}.

The transition rates, enabling functions, and probabilities of
transitions are listed in Table I and Table II.
Fig. 5. SRN model for the basic CMTS system without rejuvenation

TABLE I

<table>
<thead>
<tr>
<th>Transition</th>
<th>Firing rate</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{p\text{-aging}}$, $T_{s\text{-aging}}$</td>
<td>$\lambda_{age}$</td>
<td></td>
</tr>
<tr>
<td>$T_{p\text{-fail}}$, $T_{s\text{-fail}}$</td>
<td>$\lambda_{f} + \lambda_{Hei}$</td>
<td></td>
</tr>
<tr>
<td>$T_{p\text{-a}}$, $T_{p\text{-a, hw}}$, $T_{s\text{-a}}$, $T_{s\text{-a, hw}}$</td>
<td>$\lambda_{a}$</td>
<td></td>
</tr>
<tr>
<td>$T_{p\text{-m}}$, $T_{s\text{-m}}$</td>
<td>$\lambda_{m}$</td>
<td></td>
</tr>
<tr>
<td>$T_{s\text{-swt}}$, $T_{s\text{-swt, hw}}$, $T_{s\text{-swt, hw}}$</td>
<td>$\lambda_{swt}$</td>
<td></td>
</tr>
<tr>
<td>$T_{p\text{-reboot}}$</td>
<td>$\lambda_{rbt}$</td>
<td></td>
</tr>
<tr>
<td>$T_{p\text{-hw}}$, $T_{p\text{-hw2}}$, $T_{s\text{-hw}}$, $T_{s\text{-hw2}}$</td>
<td>$\lambda_{hw}$</td>
<td></td>
</tr>
<tr>
<td>$T_{p\text{-repair}}$, $T_{s\text{-repair}}$</td>
<td>$\lambda_{rep}$</td>
<td></td>
</tr>
<tr>
<td>$T_{p\text{-Hei}}$, $T_{s\text{-Hei}}$</td>
<td>$\lambda_{Hei}$</td>
<td></td>
</tr>
<tr>
<td>$t_1$, $t_3$</td>
<td>$c_1$</td>
<td></td>
</tr>
<tr>
<td>$t_2$, $t_4$</td>
<td>$1 - c_1$</td>
<td></td>
</tr>
</tbody>
</table>
B. SRN Model of the System with Time-based Rejuvenation

The SRN model of the system with time-based rejuvenation is shown in Figure 6. The failure and recovery characteristics of the system are the same as the basic system discussed in Section IV-A. The difference is the periodic rejuvenation performed on each node. We assume that all the nodes share the same timer and the operational PCMTS and SCMTS nodes are rejuvenated one by one when a pre-defined period of time δ has elapsed.

In Figure 6(a), when a rejuvenation starts, either transition $t_5$ or transition $t_6$ can fire. When $t_5$ fires, one token is removed from place $P_{PCMTS}$ and deposited into place $P_{rejswt}$. The switchover, rejuvenation, and return actions are represented by the path composed of $T_{swt2}$, $P_{swted}$, $P_{rejuvened}$, and $T_{swtk2}$. This describes the rejuvenation of a robust PCMTS node. Similarly, an “aged” PCMTS node can also be rejuvenated upon the firing of transition $t_6$. Only one PCMTS node can be rejuvenated at a time. This is realized by the appropriately assigned enabling functions of transitions $t_5$ and $t_6$ (Table IV). Note that the robust and “aged” nodes cannot be distinguished in this scheme. The rejuvenation of nodes in robust state is unnecessary and increases node downtime. However, this distinction could be made in a measurement-based approach in which only those “aged” nodes could be rejuvenated. After all the PCMTS nodes are rejuvenated, all the tokens in places $P_{PCMTS}$ and $P_{aged}$ are transferred to place $P_{done}$. They are returned to place $P_{PCMTS}$ through the firing of transition $t_7$ after all the operational PCMTS nodes have been rejuvenated.

Rejuvenation of the SCMTS node is represented in Figure 6(b). Similarly, the SCMTS node can be rejuvenated either from a robust state (transition $t_8$) or from an “aged” state (transition $t_9$). Firing of transition $T_{swreju}$ represents the rejuvenation action and returns the token to place $P_{SCMTS}$. Note that the rejuvenation of the SCMTS node is after the rejuvenation of those PCMTS nodes, which is realized by the enabling functions of $t_8$ and $t_9$.

The common timer is depicted in Figure 6(c). One token is initially in place $P_{timer}$ representing the start of the timer. Rejuvenation is triggered every time the token arrives at place $P_{startrey}$ and all the detected node failures have been recovered in the cluster. We approximate the deterministic interval of the timer by an r-stage Erlang distribution [17], represented by places $P_1$, $P_2$, and transitions $t_{in}$, $l_{out}$, and $T_{Er}$. The transition rate of $T_{Er}$ is set as $r/\delta$ to keep the mean of this Erlang distribution equal to the rejuvenation interval $\delta$. The accuracy of the approximation increases with the integer $r$. When all the rejuvenation operations have been accomplished, the timer is reset to zero through the firing of transition $t_{done}$. The synchronization between the rejuvenation actions and timer resetting is achieved by the enabling functions and priorities associated with transitions $t_5$, $t_6$, $t_7$, $t_8$, and $t_{done}$.

The transition rates and enabling functions of these new transitions in Figure 6 are listed in Table III and Table IV, respectively.

<table>
<thead>
<tr>
<th>Transition</th>
<th>Enabling function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{swt1}$, $T_{swt2}$</td>
<td>E1 &amp; E2</td>
</tr>
<tr>
<td>$T_{swreju}$, $T_{swfai}$, $T_{swrei}$</td>
<td>E2</td>
</tr>
<tr>
<td>$P_{PCMTS}$= 1</td>
<td></td>
</tr>
<tr>
<td>$P_{swted}$= 0 &amp; &amp; $P_{rejuvened}$= 0 &amp; &amp; $P_{rejuvened}$= 0</td>
<td></td>
</tr>
</tbody>
</table>

C. SRN Model of the System with Measurement-based Rejuvenation

The SRN model for measurement-based rejuvenation is shown in Figure 7. Since in this scheme each PCMTS/SCMTS node performs rejuvenation based on some observable system parameters rather than elapsed time, no timer is maintained in the cluster. How to choose appropriate parameters for monitoring and how to predict resource utilization trend are discussed in Section III. Here, we assume that the resource exhaustion time can be successfully predicted with probability $c_2$, which is represented by the firing of transitions $t_5$ (for a PCMTS node) and $t_7$ (for the SCMTS node). The rejuvenation of a PCMTS node requires the availability of the SCMTS node. This is guaranteed by the enabling functions of transitions $t_5$, $t_6$, $t_7$, and $t_8$ listed in Table V. With probability $1 - c_2$, a node will not be rejuvenated in the “aged” state and will eventually fail. The detection and recovery behavior is the same as discussed in the previous models.

<table>
<thead>
<tr>
<th>Transition</th>
<th>Firing rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{p_reju}$, $T_{s_reju}$</td>
<td>$\lambda_{rej}$</td>
</tr>
<tr>
<td>$T_{swt2}$, $T_{swt3}$, $T_{swtk2}$</td>
<td>$\lambda_{sw}$</td>
</tr>
<tr>
<td>$T_{Er}$</td>
<td>$\lambda_{Er}$</td>
</tr>
</tbody>
</table>

D. Availability and Cost Measures in SRN Models

As the failure of a PCMTS node does not affect the operations on the other PCMTS nodes, the CMTS cluster is still available with regard to those customers connected with the operational PCMTS nodes. The whole CMTS cluster becomes
**TABLE IV**
Enabling functions and priorities of transitions in Figure 6

<table>
<thead>
<tr>
<th>Transition</th>
<th>Enabling function</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{sw1}, T_{sw1_hw}$</td>
<td>E1 &amp; &amp; E2 &amp; &amp; E3</td>
<td></td>
</tr>
<tr>
<td>$T_{sw2}, T_{sw2_hw}, T_{sw_fail}, T_{sw_Hei}$</td>
<td>E2 &amp; &amp; E3</td>
<td></td>
</tr>
<tr>
<td>$t_5, t_6$</td>
<td>$#P_{startrej} = 1$ &amp; &amp; E1 &amp; &amp; E2 &amp; &amp; E3 &amp; &amp; E4</td>
<td>100</td>
</tr>
<tr>
<td>$t_7$</td>
<td>E3 &amp; &amp; E5</td>
<td>100</td>
</tr>
<tr>
<td>$t_8, t_9$</td>
<td>$#P_{startrej} = 1$ &amp; &amp; E2 &amp; &amp; E3 &amp; &amp; E4 &amp; &amp; E5</td>
<td>300</td>
</tr>
<tr>
<td>$t_{done}$</td>
<td>$#P_{rejuv} = 1$ &amp; &amp; E3 &amp; &amp; E5</td>
<td>200</td>
</tr>
</tbody>
</table>

E1=($#P_{SCMTS} = 1 \mid #P_{aged} = 1$)
E2=($#P_{swted} = 0$ & & $#P_{rebooted} = 0$ & & $#P_{swted\_hw} = 0$ & & $#P_{repaired} = 0$)
E3=($#P_{rejuved} = 0$ & & $#P_{swted2} = 0$ & & $#P_{rejuve1} = 0$ & & $#P_{rejuv2} = 0$)
E4=($#P_{detected} = 0$ & & $#P_{detected\_hw} = 0$)
E5=($#P_{PCMTS} = 0$ & & $#P_{aged} = 0$)
Fig. 7. SRN model for the CMTS system with measurement-based rejuvenation

<table>
<thead>
<tr>
<th>Transition</th>
<th>Enabling function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{swt}$, $T_{swt_hw}$</td>
<td>E1 &amp; E2 &amp; E3</td>
</tr>
<tr>
<td>$T_{s_hw}$, $T_{s_hw2}$, $T_{s_fail}$, $T_{s_Hei}$</td>
<td>E2 &amp; E3</td>
</tr>
<tr>
<td>$t_5$, $t_6$</td>
<td>E1 &amp; E2 &amp; E3 &amp; E3 &amp; E4</td>
</tr>
<tr>
<td>$t_7$, $t_8$</td>
<td>E2 &amp; E3 &amp; E4</td>
</tr>
</tbody>
</table>

$\text{E1} = (\#P_{SCMTS} == 1 \lor \#P_{S\_aged} == 1)$

$\text{E2} = (\#P_{s\_tuted} == 0 \&\& \#P_{rebooted} == 0 \&\& \#P_{s\_tuted\_hw} == 0 \&\& \#P_{repaired} == 0)$

$\text{E3} = (\#P_{rejuv} == 0 \&\& \#P_{s\_tuted2} == 0 \&\& \#P_{rejuved} == 0)$

$\text{E4} = (\#P_{detected} == 0 \&\& \#P_{detected\_hw} == 0)$
unavailable only when all the $N+1$ nodes fail. Based on this, system availability ($A_{sys}$) and unavailability ($U_{sys} = 1 - A_{sys}$) can be computed. We also define the capacity-oriented availability $COA$ as [17]

$$COA = \frac{N_a}{N}$$

where $N_a$ denotes the average number of available PCMTS nodes and $N$ is the total number of PCMTS nodes in the cluster.

Total number of minutes of downtime cumulated over all nodes and normalized by $N$ can be computed as $T \times (1 - COA)$, where $T$ represents the total time that a CMTS system has been executed. If the cost per time unit for one node failure is fixed at $c_f$, the expected downtime cost for the CMTS system without employing rejuvenation becomes

$$cost_{basic} = T \times (1 - COA) \times c_f.$$ 

In the system with rejuvenation, node downtime $T \times (1 - COA)$ consists of $T_f$ and $T_r$, which represent the downtime caused by node failures and rejuvenation, respectively. Then, the expected downtime cost $cost_{rej}$ is computed as

$$cost_{rej} = T_f \times c_f + T_r \times c_r$$

where $c_r$ is the cost per time unit for rejuvenation. We assume $c_r < c_f$ because rejuvenation can be scheduled so that the system cost can be kept to the minimum. In the following sections, we use the normalized downtime cost $cost_{ndc}$ to evaluate the performance improvement by applying software rejuvenation. $cost_{ndc}$ is defined as

$$cost_{ndc} = \frac{cost_{rej}}{cost_{basic}}.$$ 

To obtain the capacity-oriented availability and downtime cost measures of different system configurations, the reward rates are assigned as shown in Table VI and Table VII.

V. NUMERICAL RESULTS AND ANALYSIS

The numerical results are reported in this section. We consider a CMTS cluster with $N=7$ PCMTS nodes and 1 SCMTS node. Other parameters used in our study are summarized in Table VIII.

The two plots in Figure 8 show the expected capacity-oriented availability for time-based rejuvenation policy with different rejuvenation intervals. The left is a zoomed-in plot of the right for rejuvenation intervals less than 200 hrs. In both graphs, the rejuvenation interval $\delta$ is about 40 hrs at the intersection. This means when $\delta = 40$ hrs the system has the same capacity-oriented availability either with or without using software rejuvenation. When $\delta < 40$ hrs, the system with rejuvenation has a lower capacity-oriented availability than the system when no rejuvenation is performed. Although rejuvenating a node can return it to the aging-related fault free state, the rejuvenation should not be performed too frequently. Otherwise, the accumulation of the switchover/giveback and rejuvenation time can exceed the reduction in downtime due to node failures. The benefit of software rejuvenation is manifested when $\delta > 40$ hrs. The capacity-oriented availability is above the dotted line and reaches the maximum value when $\delta = 120$ hrs. Therefore, the optimal rejuvenation interval for the system under our study is 120 hrs. When $\delta > 120$ hrs, the capacity-oriented availability slowly decreases and the asymptotic trend is shown in the right plot. If the system is rejuvenated very infrequently such as once several months, there is almost no difference in its capacity-oriented availability as expected. This extreme situation also verifies the correctness of our analytic models.

From the viewpoint of downtime cost, the effect of the ratio $c_f/c_a$ is shown in the two plots in Figure 9. The optimal rejuvenation intervals are 120, 70, 40, and 20 hrs when the $c_f/c_a$ ratio has the value of 1, 2, 5, and 10, respectively. When the ratio increases, the optimal rejuvenation interval and the corresponding downtime cost decrease since rejuvenation becomes more cost-effective than failure. The right plot shows the asymptotic behavior in which the curves of different cost ratios all approach to the unity value.

Figure 10 shows the results for measurement-based rejuvenation policy. The capacity-oriented availability and downtime cost are shown in the left and right plots, respectively. In both plots, the higher the prediction coverage is, the better the system is, either in terms of capacity-oriented availability or downtime cost. Moreover, even when only half of the aging-related software faults can be successfully detected, the reduction in both unavailability and cost is significant. Unlike in time-based rejuvenation policy, the $c_f/c_a$ ratio has almost no impact on the downtime cost compared with the prediction coverage. Therefore, the prediction coverage is the most important parameter in measurement-based rejuvenation.

VI. CONCLUSIONS

In this paper, the software rejuvenation technique is studied in a CMTS cluster with N:1 redundancy. First, the concept of software fault classification and software rejuvenation are briefly reviewed. Second, the architecture and functionality flowcharts are provided for implementing different software

Fig. 8. Capacity-oriented availability for time-based rejuvenation
### TABLE VI
REWARD RATE FUNCTIONS FOR CAPACITY-ORIENTED AVAILABILITY

<table>
<thead>
<tr>
<th>Model</th>
<th>Reward rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic system</td>
<td>( (#P_{PCMTS} + #P_{aged} + #P_{swted} + #P_{swted_hw}) / N )</td>
</tr>
<tr>
<td>Time-based rejuvenation</td>
<td>( (#P_{PCMTS} + #P_{aged} + #P_{swted} + #P_{swted_hw} + #P_{swted_hw2}) / N )</td>
</tr>
<tr>
<td>Measurement-based rejuvenation</td>
<td>( (#P_{PCMTS} + #P_{aged} + #P_{swted} + #P_{swted_hw}) / N )</td>
</tr>
</tbody>
</table>

### TABLE VII
REWARD RATE FUNCTIONS FOR EXPECTED DOWNTIME COST

<table>
<thead>
<tr>
<th>Model</th>
<th>Reward rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic system</td>
<td>( c_f \times \left( #P_a + #P_m + #P_{detected} + #P_{rebooted} + #P_{failed_hw} + #P_{detected_hw} + #P_{repaired} \right) / N )</td>
</tr>
<tr>
<td>Time-based rejuvenation</td>
<td>( c_f \times \left( #P_a + #P_m + #P_{detected} + #P_{rebooted} + #P_{failed_hw} + #P_{detected_hw} + #P_{repaired} \right) / N + c_r \times \left( #P_{rejuv1} + #P_{rejuv2} + #P_{rejuvened} \right) / N )</td>
</tr>
<tr>
<td>Measurement-based rejuvenation</td>
<td>( c_f \times \left( #P_a + #P_m + #P_{detected} + #P_{rebooted} + #P_{failed_hw} + #P_{detected_hw} + #P_{repaired} \right) / N + c_r \times \left( #P_{rejuv} + #P_{rejuvened} \right) / N )</td>
</tr>
</tbody>
</table>

### TABLE VIII
PARAMETERS USED IN NUMERICAL STUDY

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Value (hr(^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda_{hw} )</td>
<td>Hardware failure rate</td>
<td>1/53328</td>
</tr>
<tr>
<td>( \lambda_{Hei} )</td>
<td>Software failure (caused by Heisenbugs) rate</td>
<td>1/10000</td>
</tr>
<tr>
<td>( \lambda_{age} )</td>
<td>Software aging rate</td>
<td>1/200</td>
</tr>
<tr>
<td>( \lambda_{f} )</td>
<td>Software failure rate (caused by aging-related bugs) conditioned on being in the “aged” state</td>
<td>1/200</td>
</tr>
<tr>
<td>( \lambda_{swt} )</td>
<td>Node switching rate</td>
<td>120</td>
</tr>
<tr>
<td>( \lambda_{rbl} )</td>
<td>Node reboot rate</td>
<td>10</td>
</tr>
<tr>
<td>( \lambda_{rep} )</td>
<td>Node repair rate</td>
<td>1/4</td>
</tr>
<tr>
<td>( \lambda_{a} )</td>
<td>Automatic failure detection rate</td>
<td>120</td>
</tr>
<tr>
<td>( \lambda_{m} )</td>
<td>Manual failure detection rate</td>
<td>1/4</td>
</tr>
<tr>
<td>( \lambda_{rej} )</td>
<td>Rejuvenation rate</td>
<td>60</td>
</tr>
<tr>
<td>( \delta )</td>
<td>Rejuvenation interval</td>
<td>variable</td>
</tr>
<tr>
<td>( \lambda_{Er} )</td>
<td>Transition rate in r-stage Erlang dist.</td>
<td>( r/\delta )</td>
</tr>
<tr>
<td>( r )</td>
<td>Number of stages in Erlang dist.</td>
<td>3</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>Coverage of automatic failure detection</td>
<td>0.95</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>Prediction coverage in measurement-base approach</td>
<td>variable</td>
</tr>
</tbody>
</table>
rejuvenation policies in a high availability CMTS system. Finally, the SRN models are built to study the capacity-oriented availability and downtime cost with/without deploying software rejuvenation. The numerical results from the analytic models show the significant availability improvement and downtime cost reduction when the either time-based or measurement-based rejuvenation strategy is introduced in the cluster system. For time-based rejuvenation, the optimal rejuvenation intervals have been derived so that either system unavailability or downtime cost is minimized. For measurement-based rejuvenation, the benefit from rejuvenation varies with the prediction coverage.

VII. ACKNOWLEDGEMENT

The authors from Duke University thank Kalyanaraman Vaidyanathan for helpful discussions during the development of this paper. The authors from Motorola are grateful for the valuable discussions with their colleagues Weidong Chen, Patrick Gerber, Michael Patrick and Gerard White.

REFERENCES