A proactive approach towards always-on availability in broadband cable networks

Yun Liu\textsuperscript{a,∗}, Yue Ma\textsuperscript{b}, James J. Han\textsuperscript{b}, Haim Levendel\textsuperscript{b}, Kishor S. Trivedi\textsuperscript{a}

\textsuperscript{a}CACC, Department of ECE, Duke University, P.O. Box 90291, Durham, NC 27708, USA
\textsuperscript{b}Motorola, Inc., 21440 W Lake Cook Road, 2FL/GSG, Deer Park, IL 60010, USA

Received 9 June 2003; revised 5 August 2004; accepted 19 August 2004
Available online 16 September 2004

Abstract

In this paper, we propose a high availability design of a Cable Modem Termination System (CMTS) clusters system based on the software rejuvenation technique. This proactive system maintenance technique is aimed to reduce system outages and the associated downtime cost due to the ‘software aging’ phenomenon. Different rejuvenation policies are studied from the perspectives of design, implementations, and availability assessment. To evaluate these policies, stochastic reward net models are developed and solved by Stochastic Petri Net Package (SPNP). Numerical results show that the deployment of software rejuvenation in the system leads to significant improvement in capacity-oriented availability and reduction in downtime cost. 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.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Availability; Cable modem termination system; Petri net; Preventive maintenance; Software rejuvenation; Stochastic reward net

1. Introduction

With the increasing popularity of high speed Internet services, cable modem has emerged as one of the most important 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) [27]. Customers are expecting the same high-level of availability for delivering voice, data, and video services from cable operators as provided by the traditional telephone networks.

As seen in Fig. 1, the architecture of a cable modem system comprises three major elements: an HFC cable network, a cable modem (CM) at each customer location on one end of the cable plant, and a cable modem termination system (CMTS) at the headend on the other end of the cable plant. The CMTS serves as the backbone of the HFC cable network and provides connectivity between the cable network and the Internet for both upstream and downstream traffic. In addition, the CMTS is also responsible for management services such as billing, authorization, quality of service (QoS) control, and protocol conversion. Therefore, a CMTS plays a central role in a cable modem network and involves tremendous complex hardware and software implementations. The high availability of CMTS is crucial for cable operators to provide carrier-class services to all subscribers.

Due to the importance of CMTS in cable modem systems, hardware redundancy and the corresponding resilient software features are introduced in CMTS to achieve high availability as a traditional approach. As shown in Fig. 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 Internet access service for all the customers.

∗ This research was supported by Motorola and by an AFOSR MURI grant no. F49620-1-0327.
∗ Corresponding author. Tel.: +1 919 401 0299x313; fax: +1 919 401 5589.
E-mail addresses: liu@ee.duke.edu (Y. Liu), a14438@email.mot.com (Y. Ma), cjh048@email.mot.com (J.J. Han), ail003@email.mot.com (H. Levendel), kst@ee.duke.edu (K.S. Trivedi).

0140-3664/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2004.08.022
residing in the coverage area. 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 takes over 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 the PCMTS nodes SCMTS nodes.

Deploying the fault-tolerant technique discussed above enables significantly increased system availability and reduced downtime cost. However, there exist some inherent limitations, which still 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, a CMTS cluster can hardly reach the desired availability. 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 [2,15,19,26].

Software rejuvenation in cluster systems has recently been studied in Ref. [23]. The main contribution of their work is to show how to design, implement, and evaluate software rejuvenation in a practical cluster server and how the maximum availability can be achieved by the different rejuvenation strategies and parameter settings. Our work is quite different from theirs since the CMTS cluster system in our study has completely different conceptual and practical meaning. In Ref. [23], 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 totally different (see Section 4.6). In addition, we also consider node failures caused by hardware faults and Heisenbugs (see Section 2.1) rather than only by aging-related software faults. Finally, the system recovery behavior in the presence of node failures is much more complex in CMTS due to the switchover action required for service continuity.

It is worth to mention that the software rejuvenation technology has been incorporated into the IBM Director for xSeries servers because of the system availability improvement and cost reduction [4]. 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 Sections 2 and 3, the phenomenon of software aging generally exist in almost all computer systems, especially in those systems with large and complex software. Furthermore, our proposed high availability architecture of CMTS cluster provides a convenient framework that can be easily modified to deploy different software rejuvenation strategies. The authors of this paper hope that the encouraging numerical results of the analytic models may expediate the use of software rejuvenation technology in the future high availability CMTS systems as well as other non-CMTS cluster systems.

This paper compares the availability and downtime cost of a CMTS cluster with and without software rejuvenation. Moreover, different rejuvenation policies are discussed and their performances are evaluated 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 CMTC models is simplified by using a higher-level paradigm known as stochastic reward net (SRN). Stochastic reward net models
are solved by using Stochastic Petri Net Package (SPNP) [3]. The benefits of introducing software rejuvenation are quantified from our numerical results. Furthermore, the optimization of the rejuvenation interval in the time-based rejuvenation scheme and the effect of the prediction coverage in the measurement-based rejuvenation scheme are also studied in this paper.

The rest of the paper is organized as follows. In Section 2, classification of software faults is given and the concept of software rejuvenation is reviewed. In Section 3, the outline for implementing software rejuvenation in a CMTS cluster environment is proposed. In Section 4, 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 5 and conclusions are drawn in Section 6. Appendix provides detailed description of the SRN models.

2. Software faults and software rejuvenation

2.1. 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 reuse continues, software faults are inevitable even after extensive testing and debugging. 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 [14,20].

Jim Gray has suggested classifying software faults into two categories, Bohrbugs and Heisenbugs [13]. Bohrbugs are easy to reproduce and detect. An operation containing Bohrbugs will always cause errors 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 functionality is realized through different design/implemenations. 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.

2.2. 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 [18]. 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 is triggered by some form of feedback information from the system. The rejuvenation decision is made based on current system state and/or previous system behavior, which include workload, which include workload [9], resource usage [12], and failure logs [8]. Measurement-based rejuvenation belongs to this category [24].

2.2.1. Time-based rejuvenation

This rejuvenation policy is characterized by the fact that the software is periodically rejuvenated every time a predefined time constant \( \delta \) has elapsed.

The primary challenge in a time-based rejuvenation is to determine the value of \( \delta \) so that the optimization is achieved.
in terms of minimum unavailability or downtime 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 unavailability and downtime cost when performed excessively as discussed in following sections. Therefore, choosing an proper rejuvenation interval is a critical task in a time-based rejuvenation policy.

A variant of pure time-based scheme is to consider system load with elapsed time. 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, the elapsed time \( \delta \) and the traffic load should be considered jointly [11].

2.2.2. Measurement-based rejuvenation

This rejuvenation policy is also referred to as prediction-based rejuvenation since rejuvenation decision is made based on certain monitored system parameters. Usually, these selected parameters describe the usage of some system resources such as free memory space. From the collected parameters, software aging and other anomalies can be detected through appropriate statistical techniques [4]. Furthermore, the system failure time due to resource exhaustion can be estimated by the smoothing and local regression trend detection techniques [12].

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. A manager, an agent, and a Management Information Base (MIB) are the three basic components to construct 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 degree of software aging, predict the failure time for each monitored node, and use this information to rejuvenate the whole system (or specific applications) before a software failure occurs [12].

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 failures will be detected and recovered by the existing fault-tolerant mechanism in the system.

3. An outline for implementing software rejuvenation in CMTS

3.1. Architecture and functionality flowchart

The proposed software rejuvenation technology naturally fits in a CMTS cluster with \( N + 1 \) redundancy. As mentioned before, the existing standby redundancy requires the fault detection and cluster management software, which monitors the state of each PCMTS/SCMTS node and control the switchover process when a 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 Fig. 3. The functionality of SRM and SRA varies with different rejuvenation policies such as the time-based or measurement-based schemes.

In general, the SRA is designed to respond to requests from SRM for the local resource monitoring task (Fig. 4). The SRM is responsible for making the decision as to which node needs rejuvenation. For the SCMTS node, it can be rejuvenated whenever appropriate, 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 CMTS node may have different granularity, either restarting the whole operating system or restarting a specific application program.

3.2. Design for time-based policy

In this policy, a rejuvenation is triggered after a fixed time has elapsed. If the timer is maintained by the SRM, no SRA is needed to implement rejuvenation. Otherwise, the SRM can obtain the timing information from each node reported by the SRA. When a predefined time interval has elapsed, a rejuvenation trigger is generated by the SRM and then 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 [9], simulation model, or
previous failure data from the field [8]. SRM can also maintain a calendar in which the system administrator can specify the days when a rejuvenation is allowed and when it is forbidden [4]. This provides more flexibility in resolving potential conflicts under system update or reconfiguration.

3.3. Design for measurement-based policy

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

According to the study in Ref. [12], 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 the I/O devices can also be included. Seasonal Kendall test was proposed in Ref. [12] 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 h 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 $S = \sum_{i=1}^{n} \sum_{j=k+1}^{n} \text{sgn}(x_i - x_j)$, 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 a rejuvenation is carried out before the predicted time to resource exhaustion, the likelihood of software crash or hang will be reduced.

4. Modeling CMTS with software rejuvenation

4.1. Introduction to SRN

Stochastic reward net (SRN) [5] is an extension of Petri net (PN) [17]. As a high-level description language for formal specification of complex systems, SRN has been widely used in the area of performance and dependability analysis due to its conciseness and clarity in visual and conceptual presentation. As a result, they allow the designer to focus more on the system characteristics being modeled rather than on the error-prone specification of system state space.

A PN is a bipartite directed graph with two types of nodes: places and transitions. Each place may contain an arbitrary (natural) number of tokens. Each transition may have zero or more input arcs, coming from its input places; and zero or more output arcs, going to its output places. A transition is enabled if all of its input places have at least as many tokens as required by the multiplicities of the corresponding input arcs. When enabled, a transition can fire and will remove from each input place and add to each output place the number of tokens corresponding to the multiplicities of the input/output arcs. A marking depicts the state of a PN which is characterized by the assignment of tokens in all the places. With respect to a given initial marking, the reachability set is defined as the set of all markings reachable through any possible firing sequences of transitions, starting from the initial marking. For a graphical presentation, places are depicted as circles, transitions as bars, tokens as dots or integers in the places, and arcs as arrows.

Generalized stochastic Petri nets (GSPNs) [1] extend the PNs by assigning a firing time to each transition. Transitions with exponentially distributed firing times are called timed transitions while the transitions with zero
firing times are called immediate transitions. A marking in a GSPN is called vanishing if at least one immediate transition is enabled; otherwise it is called a tangible marking. For a given GSPN, an extended reachability graph (ERG) is generated with the markings of the reachability set as the nodes and some stochastic information attached to the arcs, thus connecting the markings to each other. Under the condition that only a finite number of transitions can fire in finite time with non-zero probability, it can be shown that a given ERG can be reduced to a homogeneous continuous time Markov chain \[1\].

In order to make more compact models of complex systems, several extensions are made to GSPN, leading to the SRN. One of the most important features of SRN is its ability to allow extensive marking dependency. In an SRN, each tangible marking can be assigned with one or more reward rates. Parameters such as the firing rate of the timed transitions, the multiplicities of input/output arcs and the reward rate in a marking can be specified as functions of the number of tokens in any place in the SRN. Another important characteristic of SRN is the ability to express complex enabling/disabling conditions through guard functions. This can greatly simplify the graphical representations of complex systems. For an SRN, all the output measures are expressed in terms of the expected values of the reward rate functions. To get the performance and reliability/availability measures of a system, appropriate reward rates are assigned to its SRN. As an SRN can be automatically transformed into a Markov reward model (MRM) \[5,22\], steady state and/or transient analysis of the MRM produces the required measures of the original SRN. In this paper, we use the tool SPNP \[6,7\] to specify and solve the SRN models.

4.2. Modeling assumptions

In the following sections, we construct SRN models for different system configurations and also provide the availability and cost measures for each case in terms of reward rate specifications. The assumptions for the models under study are given as follows:

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

(2) Hardware failures can always be detected and then repaired. 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 (100 \(c_1\) percent) manifest themselves as node level crashes or not being able to respond to heartbeat messages and thus can be automatically detected by the embedded system software in a short period of time. However, there always exists a small portion (100(1 \(-c_1\)) percent) of software failures which cannot be revealed in an automatic manner. They usually require a substantially longer time to detect with human intervention such as customer complaints or scheduled system maintenance. We call the first case automatic detection and the second case manual detection. Therefore, \(c_1\) can be regarded as the coverage of automatic detection as shown in Tables 3 and A1.

(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. Therefore, the conditional probability of multiple components failure is small enough to be ignored.

(5) HIS is designed to be a fail-safe device. In other words, its failure will not have any impact on the network elements not involved in the switchover procedure and hence the active traffic will not be affected. This requirement could be achieved in reality because of the relative simplicity with regard to HIS functionality in the system as discussed in Section 1.

(6) The distribution of time between hardware failures and software failures caused by Heisenbugs are assumed to be exponential. Time between software failures caused by aging-related faults is assumed to be hypo-exponentially distributed as is done by several other authors \[10,15\]. Failure detection time and node switchover/reboot/rejuvenation/giveback time are assumed to be exponentially distributed.

4.3. SRN model of a basic system without rejuvenation

Fig. 5 shows the SRN model of the basic system that has not employed the software rejuvenation technique. In this model, a PCMTS (see submodel (a)) and SCMTS (see submodel (b)) have the same failure behavior, i.e. they may fail due to either hardware failures or software failures. In the latter case, failures may be caused by either Heisenbugs or aging-related bugs. The two submodels differ in their recovery behaviors. Whenever a PCMTS needs to repair/reboot, it has to switchover the ongoing tasks to the SCMTS. However, the repair action of the SCMTS is carried out without considering the state of any PCMTS. This dependency is reflected in the enabling functions and priorities of various transitions as explained in detail in Appendix.
4.4. SRN model of the system with time-based rejuvenation

The SRN model of the system with the time-based rejuvenation is shown in Fig. 6. The failure and recovery characteristics of the system are the same as the basic system discussed in Section 4.3. The difference in submodel (a) and (b) is due to the periodic rejuvenation performed on each node, which characterizes the time-based rejuvenation policy. Submodel (c) is added to model the deterministic rejuvenation timer. We assume that all the nodes share the same timer and the operational PCMTS and SCMTS nodes are rejuvenated one by one when a predefined period of time $\delta$ has elapsed. A detailed description about the interaction among the three submodels is provided in Appendix.

4.5. SRN model of the system with measurement-based rejuvenation

The SRN model for the measurement-based rejuvenation is shown in Fig. 7. Since in this scheme each PCMTS/SCMTS node performs rejuvenation based on some observable system parameters rather than elapsed time, timer is not maintained in the cluster. This explains why the model consists of two submodels as in the basic system. We assume that the resource exhaustion time can be successfully predicted with a specific probability, which is $c_2$ as can be seen in Appendix. This results in a different rejuvenation behavior. For either the PCMTS or SCMTS, rejuvenation is only triggered from an ‘aged’ state rather than both the robust and ‘aged’ states as in the time-based rejuvenation scheme.
4.6. Availability and cost measures in SRN models

As the failure of a PCMTS node does not affect the operations on other PCMTS nodes, the CMTS cluster is still available with regard to those customers connected with other operational PCMTS nodes. The whole CMTS cluster becomes unavailable only when all the \( N + 1 \) nodes fail. Although system availability \( (A_{sys}) \) can be computed according to this definition, we choose capacity-oriented availability (COA) \([21]\) as our availability measure since it reflects the perception from customers

\[
\text{COA} = \frac{N_a}{N},
\]

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

Total accumulated downtime over all nodes and normalized by \( N \) is computed as \( T(1 - \text{COA}) \), where \( T \) represents the total time that a CMTS system has been in operation. 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

\[
\text{cost}_{\text{basic}} = T(1 - \text{COA})c_f.
\]

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

\[
\text{cost}_{\text{rej}} = T_f c_f + T_r c_r,
\]

where \( c_r \) is the cost per time unit for rejuvenation. We assume \( c_r < c_f \) because rejuvenations are planned and scheduled in advance so that the system cost can be kept to the minimum. In the following sections, we use the normalized downtime cost \( \text{cost}_{\text{ndc}} \) to evaluate the performance improvement by applying software rejuvenation. \( \text{cost}_{\text{ndc}} \) is defined as

\[
\text{cost}_{\text{ndc}} = \frac{\text{cost}_{\text{rej}}}{\text{cost}_{\text{basic}}}. \]

To obtain the capacity-oriented availability and downtime cost measures of different system configurations, the reward rates are assigned as shown in Tables 1 and 2.
5. Numerical results and analysis

In this section, we examine the capacity-oriented availability, normalized downtime cost, and the impact of crucial rejuvenation parameters on these two measures for our SRN models. For each model, we consider a high availability CMTS cluster with $N=7$ PCMTS nodes and one SCMTS node. Table 3 summarizes other parameters used in our models. The numerical results are obtained through the software tool Stochastic Petri Net Package (SPNP) developed by Duke researchers [3]. Table 4 shows the state space for the SRN models with the parameters listed in Table 3.

The two plots in Fig. 8 shows the expected capacity-oriented availability for the time-based rejuvenation policy with different rejuvenation intervals. The left is a zoomed-in plot of the right for rejuvenation intervals less than 200 h. In both graphs, the rejuvenation interval $\delta$ is about 40 h at the intersection of the dotted line (w/o rejuvenation) and the solid line (w/rejuvenation). This mean when $\delta=40$ h the CMTS cluster system has the same COA either with or without using software rejuvenation. When $\delta<40$ h, the system with rejuvenation has a lower COA than the system, where 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 $\delta>40$ h. The COA is above the dotted line and reaches the maximum value when $\delta=120$ h. Therefore, the optimal rejuvenation interval for the system under our

![Diagram](image)

(a) PCMTS

![Diagram](image)

(b) SCMTS

Fig. 7. SRN model for the CMTS system with measurement-based rejuvenation.

Table 1

<table>
<thead>
<tr>
<th>Model</th>
<th>Reward rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic system</td>
<td>$(#P_{PCMTS} + #P_{aged} + #P_{swt2} + #P_{swt,low})/N$</td>
</tr>
<tr>
<td>Time-based rejuvenation</td>
<td>$(#P_{PCMTS} + #P_{aged} + #P_{swt2} + #P_{swt,low} + #P_{swt,low})/N$</td>
</tr>
<tr>
<td>Measurement-based</td>
<td></td>
</tr>
<tr>
<td>rejuvenation</td>
<td>$(#P_{PCMTS} + #P_{aged} + #P_{swt2} + #P_{swt,low})/N$</td>
</tr>
</tbody>
</table>
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 the time-based rejuvenation policy, the ratio \( c_1/c_r \) has much less significant impact on the downtime cost compared with the prediction coverage. Therefore, the prediction coverage is the most important parameter in the measurement-based rejuvenation and should be the focus for improvement.

More insights about the impact of rejuvenation parameters on system availability and downtime cost can be seen from the downtime composition as shown in Fig. 11. In the time-based rejuvenation scheme (see left plot of Fig. 11), when the rejuvenation interval decreases, the downtime caused by rejuvenation increases monotonously while the downtime due to failures decreases monotonously. When the interval is less than a certain threshold, the increase in downtime caused by rejuvenation exceeds the reduction in the downtime due to failures and thereby the system availability could deteriorate with over-frequent rejuvenation. Obviously, there exists a unique optimal rejuvenation interval to minimize either the total downtime or the total downtime cost with different cost ratios. In the measurement-based rejuvenation (see right plot of Fig. 11), rejuvenation is always preferable to failures as long as the assumption that performing a rejuvenation incurs less downtime and cost holds. This is why we observe that the downtime due to failures decreases faster than the downtime caused by failures when the prediction coverage increases.

### 6. Conclusions

In this paper, the software rejuvenation technique is systematically studied in a CMTS cluster with N:1 redundancy. First, the concept of software fault classification and software rejuvenation is briefly reviewed. Second, the architecture and functionality flowcharts are provided for implementing different software rejuvenation policies in a high availability CMTS system. Finally, the SRN models are built to evaluate the capacity-oriented availability and downtime cost with/without deploying software rejuvenation. The numerical result from the analytic models show the significant availability improvement and downtime cost reduction when either the time-based or measurement-based rejuvenation strategy is introduced in the cluster system. For the time-based rejuvenation, the optimal rejuvenation intervals have been

---

**Table 2**

<table>
<thead>
<tr>
<th>Model</th>
<th>Reward rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic system</td>
<td>( c_1(P_{rejuv1} + P_{rejuv2}) + \sum P_{failed}_hw + \sum P_{rebooted}_hw + \sum P_{detected}_hw + \sum P_{detected}_hw + \sum P_{restored}_hw) ) /( N )</td>
</tr>
<tr>
<td>Time-based rejuvenation</td>
<td>( c_1(P_{rejuv1} + P_{rejuv2}) + \sum P_{failed}_hw + \sum P_{rebooted}_hw + \sum P_{detected}_hw + \sum P_{detected}_hw + \sum P_{restored}_hw) ) /( N )</td>
</tr>
<tr>
<td>Measurement-based rejuvenation</td>
<td>( c_1(P_{rejuv1} + P_{rejuv2}) + \sum P_{failed}_hw + \sum P_{rebooted}_hw + \sum P_{detected}_hw + \sum P_{detected}_hw + \sum P_{restored}_hw) ) /( N )</td>
</tr>
</tbody>
</table>

**Table 3**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Value (h(^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda_{hw} )</td>
<td>Hardware failure</td>
<td>1/53,328 [16]</td>
</tr>
<tr>
<td>( \lambda_{Hei} )</td>
<td>Software failure (caused by Heisenbugs) rate</td>
<td>1/10,000</td>
</tr>
<tr>
<td>( \lambda_{ag} )</td>
<td>Software aging rate</td>
<td>1/200</td>
</tr>
<tr>
<td>( \lambda_{s} )</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_{rej} )</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_{rez} )</td>
<td>Rejuvenation rate</td>
<td>60</td>
</tr>
<tr>
<td>( \delta )</td>
<td>Rejuvenation interval</td>
<td>Variable</td>
</tr>
<tr>
<td>( \lambda_{tr} )</td>
<td>Transition rate in r-stage Erlang dist.</td>
<td>Variable</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-based approach</td>
<td>Variable</td>
</tr>
</tbody>
</table>

**Table 4**

<table>
<thead>
<tr>
<th>Model</th>
<th>No. of tangible markings</th>
<th>No. of nonzero transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic model</td>
<td>21,120</td>
<td>144,540</td>
</tr>
<tr>
<td>Time-based rejuvenation model</td>
<td>188,790</td>
<td>1,448,395</td>
</tr>
<tr>
<td>Measurement-based rejuvenation model</td>
<td>172,788</td>
<td>1,232,352</td>
</tr>
</tbody>
</table>
Fig. 8. Capacity-oriented availability for time-based rejuvenation.

Fig. 9. Normalized downtime cost for time-based rejuvenation.

Fig. 10. Capacity-oriented availability and normalized downtime cost for measurement-based rejuvenation.

Fig. 11. Normalized downtime composition in time-based and measurement-based rejuvenation.
found so that either system unavailability or downtime cost can be minimized. For the measurement-based rejuvenation, the relationship between prediction coverage and availability is revealed and discussed.

Acknowledgements

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, Paul Piekarski and Gerard White.

Appendix A

A.1. Description of the SRN model of a basic system without rejuvenation

In Section 4.3, Fig. 5 shows the SRN model of the basic system that has not employed software rejuvenation technique. Initially, N tokens are in place $P_{PCMTS}$ in Fig. 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 impossible for the system to encounter aging-related faults. However, failures can be caused by Heisenbugs or hardware faults, represented by the firing of transitions $T_{p\_he}^f$ and $T_{p\_hw}^f$. As time progresses, transition $T_{p\_aging}$ fires and removes one token from place $P_{PCMTS}$ and deposits one token in place $P_{aged}$. This indicates that a PCMTS node eventually enters the 'software aging' state and the node may fail due to resource exhaustion. Note that the PCMTS node is still operational in this state but its performance might be degraded. Firing of transitions $T_{p\_fail}$ 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 $c_1$ the software failure is automatically detected, which is represented by the path composed of transition $t_3$, place $P_a$, and transition $T_{p\_a}$. Or, with probability $1 - c_1$, the software failure can only be detected manually, which is represented by the path composed of transition $t_4$, 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 failures. The SCMTS node has a similar failure behavior as shown in Fig. 5(b).

Firing of transition $T_{s\_swt}$ in Fig. 5(a) represents the switchover action, where the SCMTS node takes over all the running operations of a failed PCMTS node. Transition $T_{s\_swt}$ 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 failures. Firing of transition $T_{p\_reboot}$ 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 operations 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 $T_{c\_swt}$.

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 recovery 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 automatic 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 Tables A1 and A2.

A.2. Description of the SRN model of the system with time-based rejuvenation

In Section 4.4, the SRN model of the system with the time-based rejuvenation is shown in Fig. 6. In Fig. 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 $P_{rejuv}$. The switchover, rejuvenation, and return actions are represented by the path composed of $T_{swt2}$, $P_{swted2}$, $T_{p\_rejuv}$, $P_{rejuved}$, and $T_{c\_swt}$.

Table A1

<table>
<thead>
<tr>
<th>Transition</th>
<th>Firing rate</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{p_aging}$, $T_{s_aging}$</td>
<td>$\lambda_{age}$</td>
<td></td>
</tr>
<tr>
<td>$T_{p_fail}$, $T_{s_fail}$</td>
<td>$\lambda_1 + \lambda_{Hei}$</td>
<td></td>
</tr>
<tr>
<td>$T_{p_a_hw}$, $T_{s_a_hw}$</td>
<td>$\lambda_a$</td>
<td></td>
</tr>
<tr>
<td>$T_{p_m}$, $T_{s_m}$</td>
<td>$\lambda_m$</td>
<td></td>
</tr>
<tr>
<td>$T_{s_swt}$, $T_{s_swt_hw}$, $T_{c_swt}$</td>
<td>$\lambda_{swt}$</td>
<td></td>
</tr>
<tr>
<td>$T_{swt_hw}$</td>
<td>$\lambda_{swt_hw}$</td>
<td></td>
</tr>
<tr>
<td>$T_{s_reboot}$</td>
<td>$\lambda_{swt_reboot}$</td>
<td></td>
</tr>
<tr>
<td>$T_{s_he}$, $T_{s_Hei}$</td>
<td>$\lambda_{Hei}$</td>
<td></td>
</tr>
<tr>
<td>$t_5$, $t_3$</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>

Table A2

<table>
<thead>
<tr>
<th>Transition</th>
<th>Enabling function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{s_swt}$, $T_{c_swt}$</td>
<td>$E_1$ &amp; $E_2$</td>
</tr>
<tr>
<td>$T_{s_he}$, $T_{s_hw2}$, $T_{s_fail}$, $T_{s_Hei}$</td>
<td>$E_2$</td>
</tr>
</tbody>
</table>

$E_1 = (# P_{SCMTS} = 1) && (# P_{s\_aged} = 1)$. $E_2 = (# P_{swted} = 0) && (# P_{rebooted} = 0) && (# P_{swted\_hw} = 0) && (# P_{repaired} = 0)$. 
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 $t_5$ and $t_6$ (Table A4). 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 Fig. 6(b). Similarly, the SCMTS node can be rejuvenated either from a robust state (transition $t_5$) or from an ‘aged’ state (transition $t_6$). Firing of transition $T_{s\_rejuv}$ 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 Fig. 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 $P_{startrej}$ 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 [21], represented by places $P_1$, $P_2$, and transitions $l_{in}$, $l_{out}$, and $T_{E_r}$. The transition rate of $T_{E_r}$ 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 Fig. 6 are listed in Tables A3 and A4, respectively.

### Table A3

<table>
<thead>
<tr>
<th>Transition</th>
<th>Firing rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{s_rejuv}$</td>
<td>$\lambda_{rejuv}$</td>
</tr>
<tr>
<td>$T_{s_rejuv}$</td>
<td>$\lambda_{rejuv}$</td>
</tr>
<tr>
<td>$T_{out2}$, $T_{out3}$</td>
<td>$\lambda_{out}$</td>
</tr>
<tr>
<td>$T_{E_r}$</td>
<td>$\lambda_{E_r}$</td>
</tr>
</tbody>
</table>

### Table A4

<table>
<thead>
<tr>
<th>Transition</th>
<th>Enabling function</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{out}$, $T_{out_hw}$</td>
<td>$E_1$ &amp; $E_2$ &amp; $E_3$</td>
<td></td>
</tr>
<tr>
<td>$T_{l_{in}}$, $T_{l_{out2}}$, $T_{s_heal}$, $T_{s_hw}$</td>
<td>$E_2$ &amp; $E_3$</td>
<td></td>
</tr>
<tr>
<td>$t_5$, $t_6$</td>
<td>$s_{startrej} = 1$ &amp; $E_1$ &amp; $E_2$ &amp; $E_3$ &amp; $E_4$ &amp; $E_5$</td>
<td>100</td>
</tr>
<tr>
<td>$t_7$</td>
<td>$E_3$ &amp; $E_5$</td>
<td>100</td>
</tr>
<tr>
<td>$t_8$, $t_9$</td>
<td>$s_{startrej} = 1$ &amp; $E_2$ &amp; $E_3$ &amp; $E_4$ &amp; $E_5$</td>
<td>300</td>
</tr>
<tr>
<td>$t_{done}$</td>
<td>$s_{rejuv} = 1$ &amp; $E_3$ &amp; $E_4$ &amp; $E_5$</td>
<td>200</td>
</tr>
</tbody>
</table>

### Table A5

<table>
<thead>
<tr>
<th>Transition</th>
<th>Enabling function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{out}$, $T_{out_hw}$</td>
<td>$E_1$ &amp; $E_2$ &amp; $E_3$</td>
</tr>
<tr>
<td>$T_{l_{in}}$, $T_{l_{out2}}$, $T_{s_heal}$, $T_{s_hw}$</td>
<td>$E_2$ &amp; $E_3$</td>
</tr>
<tr>
<td>$t_5$, $t_6$</td>
<td>$E_1$ &amp; $E_2$ &amp; $E_3$ &amp; $E_4$</td>
</tr>
<tr>
<td>$t_7$, $t_8$</td>
<td>$E_2$ &amp; $E_3$ &amp; $E_4$</td>
</tr>
</tbody>
</table>

### References


Yun Liu is a PhD student at Duke University. He received his MS degree in 2002 from the Department of Electrical and Computer Engineering, Duke University, Durham, NC. His research interests include high availability system analysis and performance and survivability assessment of telecommunications networks.

Yue Ma is a Product Manager at Motorola. His areas of interest are in the availability, design, performance and reliability analysis of wireless/broadband access networks, computing systems and mission critical systems. He has published more than 20 technical papers in journals and conferences. Ma holds a PhD and an MS in Computer Science from Duke University, where he was elected Phi Beta Kappa, and a Cand. Scient. and a Cand. Mag. in Informatics from University of Oslo.

James J. Han is director of Systems Engineering Technology Center in Motorola’s Global Software Group. He received his BS from Xian Jiaotong University, China, MS in applied mathematics and PhD in electrical engineering from the Ohio State University. After teaching electrical and computer engineering in Southern Illinois University and Illinois Institute of Technology as professors for 6 years, he joined AT&T Bell laboratories and worked on communication system architecture, performance, and software development. He is director of Systems Engineering Technology Center in Motorola’s Global Software Group, and working on R&D projects in wireless communication systems. His research interests are on availability/capacity/performance of next generation wireless and broadband communication systems. He has published more than 100 technical papers in journals and conferences. He is a senior member of IEEE.

Haim Levendel was director of High Reliability and Availability Technology Center, Motorola when he retired in 2003. He held leadership positions in all product development phases from front-end to factory and customer support, created partnerships inside and outside the company, and achieved national and international recognition in hardware and software quality assessment and prediction, fault tolerance, dependability and software technologies.

Kishor S. Trivedi is the Hudson Professor in the Department of Electrical and Computer Engineering at Duke University, Durham, NC. He is the author of the well-known text Probability and Statistics with Reliability, Queuing and Computer Science Applications. He has recently published two books: Performance and Reliability Analysis of Computer Systems and Queueing Networks and Markov Chains. His Research interests are in reliability and performance assessment of computer and communication systems. He is an IEEE Fellow, and a Golden Core Member of IEEE Computer Society.