Improved Reliability of Large Scale Publish/Subscribe based MOMs using Model Checking

Yue Jia, Eliane Bodanese, Chris Phillips, John Bigham, Ran Tao
Department of Electronic Engineer and Computer Science
Queen Mary, University of London
London, United Kingdom
Email: {yue.jia, eliane.bodanese, chris.phillips, john.bigham, ran.tao@eecs.qmul.ac.uk}

Abstract— **Many software systems operate across different geographically distributed hardware platforms, operating systems and programming languages. Publish/subscribe based Message Oriented Middleware (MOM) provides loose coupling and an efficient, asynchronous and scalable way of communication. However, as the complexity of such systems increase, manual verification of reconfiguration policies becomes unrealistic. The task calls for automated means of proof-checking configuration information in order to improve the reliability of large-scale MOM systems. This paper proposes a new model checking approach with temporal logic specifications to design and verify a system configuration. Model checking is a powerful technique, however the creation of appropriate finite state models for the systems being checked are complex and difficult to use in practice by non-formalists. The research presented in this paper finds suitable abstractions that reduce the system to a finite state model. The tools we developed for the generation of such models can be easily used by non-formalists. The systems models created using our techniques manages state explosion thanks to the choices of our abstractions. An example of the use of our tools and techniques is presented for a 50 node MOM, where the reachability of all topics and the presence of loops are proof-checked.**

Index Terms—**message oriented middleware, model checking, large scale system, publish/subscribe**

I. INTRODUCTION

A synchronous system is characterized by tight coupling, which requires both the caller and the callee to be available at the same time. This type of system raises considerable challenges when trying to implement certain applications or manage the interaction between clients and servers in terms of fault tolerance and availability. Asynchronous systems relax this tight coupling constraint and are well suited to messaging applications. An asynchronous system employs queues to store messages, and can guarantee that messages are retained even after failures arise. The advantages over synchronous solutions are that: application recipients do not need to be "online" at the time a message is sent; and queues facilitate communication across heterogeneous networks and systems while still being able to make some assumptions about the behavior of the message handling. Message Oriented Middleware (MOM) is a loosely coupled asynchronous framework focused on sending and receiving messages over distributed heterogeneous platforms [1].

Existing MOMs fall into one of two categories: enterprise messaging systems and real-time messaging systems. With the intention of addressing traditional business needs, enterprise messaging systems provide message delivery assurance and transactional guarantees. They usually implement the Java Message Service (JMS) standard [2] and can transport messages over a wide area across multiple domains. However, they do not proactively manage messaging performance. As such, applications cannot predict or depend on when messages will arrive at the destination. Real-time messaging systems, on the other hand, offer QoS assurance by allocating resources and scheduling messages based on application-specific QoS objectives. They often conform to the Data Distribution Service (DDS) standard [3].

Message Oriented Middleware adopts a message centric approach and usually employs both message queuing and publish/subscribe communication schemes. Figure 1 shows an example of a MOM system based on a publish/subscribe structure. Brokers are inter-connected through an overlay network where application components attach to a local broker [1] and they do not interact directly. Instead, their communications are mediated by an additional logical layer, called a dispatcher. In a topic based publish/subscribe MOM system, each message is classified as belonging to one of a fixed set of topics. A publisher labels each message it produces with a particular topic. Similarly, the subscriber has the ability to express their interest in a topic a pattern of topics to a broker, and these are collected by the dispatcher in a suitable data structure. When a component publishes a message, the dispatcher matches this against existing subscriptions, and delivers the message to all those application components that issued matching subscriptions. This process is usually referred to as message filtering. There can be an arbitrary number of topics in the system. Each endpoint can publish and subscribe to one or many topics, while each broker can

978-1-4799-0913-1/14/$31.00 ©2014 IEEE

perform publish/subscribe matching, transport messages to local endpoints or neighboring brokers, and optionally perform message mediation.

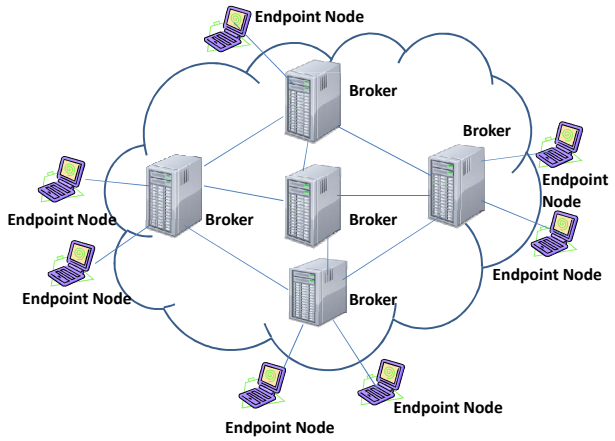

Figure 1: Example Publish-Subscribe System Model

Using this style of interaction, the sender does not know the identity of the receivers: it is the dispatcher inside the broker that identifies them dynamically. As a consequence, new components can join a federation, become immediately active, and cooperate with others without requiring any reconfiguration of the architecture. Due to this flexible structure, MOMs can deliver a service that allows content providers and consumers to concentrate on the production and consumption of transmitted information. The key advantage of a MOM architecture is that it reduces the number of point-to-point connections that need to be managed by the applications in a complex business-critical IT system.

Recent studies show that configuration of the network access control is one of the most complex and error prone network management tasks [12]. For this reason, network misconfiguration has become the main source of network unreachablility and vulnerability problems. To achieve resilience, an efficient way to configure and reconfigure the system is necessary. The main goal of configuring and reconfiguring systems is to make sure that the system continues to operate normally. However, if the protocols which perform the configuration and reconfiguration have flaws themselves, the configuration and reconfiguration will be prone to errors. It is therefore important to devise a means of verifying if the configuration is correct. Publish/subscribe systems are often complex and hard to test. In particular, given the inherent non-determinism in the order of event receipts, delays in event delivery, and variability in the timing of event announcements, the number of possible system executions consequently become combinatorially large. It is relatively easy to configure an individual device such as a broker or a firewall, but it is extremely complex to attempt a consistent configuration of a large network with many connections between devices. So this paper focuses on building a MOM system model and verifying the reachability and configuration rules of a modeled system.

The organization of the remaining of this paper is as follows. In section II we present an overview of model checking techniques and a brief introduction of basic concepts of temporal logic. Section III considers the relevant related work. In Section IV we present how a MOM model is created, the abstractions made for the verification of the reachability of topics and loop detection, and the design choices for path detection. Section V describes the procedure of building a MOM model in NuSMV. Section VI shows an example of the verification and detection of failures in paths and brokers in 50 node MOM. Section VII concludes with a brief discussion of the addition of new constraints to be verified in our MOM model.

II. An Overview of Model Checking and Temporal Logic

Model checking is an automatic technique for verifying a finite-state system, such as sequential circuit designs and communication protocols. Formulas are written in temporal logic, and a reactive system can be modeled as a state transition graph. An efficient search procedure is used to determine whether or not the state transition graph satisfies the formulas [4].

There are several advantages in using a model checking technique. The most important one is that the procedure of verifying is highly automated. Typically, the user provides a high-level abstraction of the model and the formulas that are to be checked against. The model checker will either terminate with the answer ‘true’, indicating that the model satisfies the formula, or it will give a counter example execution that shows why and where the formula failed. This is typically achieved via extensive simulations and searched every possible state. The counter example helps to detect potential problems in the system or the violation of the protocol.

However, previous model checkers were only able to check small systems and protocols [5, 6, 7]. They were not capable of operating on large systems because of the state explosion problem [8]. With the help of an Ordered Binary Decision Diagram (OBDD) the capability of a model checker can be increased dramatically [9]. OBDD can effectively reduce useless states. So the number of nodes in an OBDD no longer directly depends on the actual number of states or the extent of the translation relations. With this accomplishment, a number of major companies including Intel, Motorola, Fujitsu and AT&T have started using symbolic model checkers to verify actual circuits and protocols [4].

The combination of a model checking algorithm together with a representation of the translation relation (using OBDD), is called Symbolic Model Checking.

NuSMV is a symbolic model checker developed by ITC-IRST and UniTN with the collaboration of CMU and UniGE [10]. NuSMV provides a language for describing models a user wishes to verify and it directly checks the validity of Linear-time Temporal Logic (LTL) or Computation Tree Logic (CTL) specifications on those models. Statements are expressed in temporal logic that are used for reasoning about things that change over time.

A reactive system can be modeled as a state transition graph [11]. It provides as output either the word 'true', when the rules/specifications hold, or a trace showing why one of the rules or specifications does not hold true [12].

Typically, in CTL temporal connectives are a pair of symbols. The first part is composed of ***A*** or ***E***. ***A*** means 'along all paths' and ***E*** means 'along at least one path'. The second part of the pair is composed of ***X***, ***F***, ***G***, or ***U***, meaning 'next state', 'some future state', 'all future states' and 'until', respectively. For example, ***AF p*** means for all paths p will finally be true and ***EF p*** means there is at least one path in which p will finally be true. By using temporal logic, conditions that change over time can be represented, for example, 'I will be hungry eventually' [13].

III. RELATED WORK

Quite a few attempts have been applied to develop formal foundations for specifying and testing publish/subscribe systems [14, 15, 16, 17, 18], and this area remains a fertile one for formal verification. While these papers provide a formal method for testing publish/subscribe systems, at present they require an expert in formal specification and theorem proving to use them effectively. Our research bases on those researches, but aims to have more accessible and specific features of a publish/subscribe system to be checked by NuSMV model checkers. Unfortunately, existing notations and methods are difficult to use in practice by non-formalists, and require considerable proof machinery to carry out.

In [19], the authors propose a model checking for publish/subscribe architectures that provides a set of pluggable modules that allow the modeler to choose one possible design of a publish/subscribe system out of a set of choices. The authors believed that a typical system could be divided into the following elements: the components which encapsulate data and functionality; the event types that indicate the events that can be announced; the shared variables and bindings between events; event delivery policy and the concurrency model. Thus, in [19], the research built those elements into different reusable entities to constitute a model of publish/subscribe system.

However, available models in [19] are far from capable of verifying the different characteristics of existing publish/subscribe systems. For instance, the message dispatching mechanism is only characterized in terms of delivery policy (which is asynchronous, synchronous, immediate or delayed), and application entities cannot change their subscriptions at run-time. Those are extended in [20] by adding more expressive events, dynamic delivery policies and dynamic event method bindings. Then the paper [21] applied these features to represent a transformational framework that starts from producing Extensible Markup Language (XML) data for model checking as well as executable artifacts for testing. But the paper [21] only deals with the specification of different delivery policies depending on the overall state of the model, and still does not capture real-time constraints. Researchers in [21] tailored SPIN, one of the model checking tools, into a new one named 'Bogor' and added Bogor into a Java development environment. However, as with SPIN, Bogor does not support CTL and time constraints. So the work proposed in this paper uses NuSMV rather than Bogor to support CTL in the MOM verification process.

Rather than building a generalized model [19, 20, 21] for publish/subscribe systems that leave many of details out, this paper focuses on verifying configuration and reconfiguration for a PS overlay network using a NuSMV code generator to simplify processes of building and checking the model.

E. Al-Shaer et al. [12] presented a novel approach for modeling the global end-to-end behavior of an access control configuration for an entire network. The model represents a network as a finite state machine where the header and location information of the message which belonging to a topic determine the state. Furthermore, the message header information determines the whole transitions for a message. For a message, the broker which publics the message is the source, the broker which subscribes to this message is the destination. Inside the message header there is information giving the message source IP address, the destination address, and its current location. The rules for each device are also modeled.

32-bits are used for the source IP address, the destination IP address, and the device currently processing the packet, with 16-bit source port number and destination port numbers in the basic network model of [12]. In order to illustrate this approach, an example containing only 2 bits for the source IP, destination IP, and location IP, and 1 bit for the source port and destination port is given. The formulas use s_1, d_1, l_1 for the higher order bit in the source IP address, the destination IP address, and the location of interested IP address respectively. s_0, d_0, l_0 are used for the lower order bits. $s'_0, d'_0, s'_1, d'_1 \ldots$ represent the values of the bits in the next state with the same interpretation as the unprimed versions above.

Assume a broker with IP address 3 sends all messages in different topics destined for IP addresses 1 and 0 to IP address 0 (next hop), while all other topics are sent to IP address 2 as a default gateway.

The policy described above can be formulated as:

$$(\overline{d_1} \wedge \overline{l'_1} \wedge \overline{l'_0}) \vee (d_1 \wedge l'_1 \wedge l'_0) \qquad (1)$$

This formula (1) shows two possible situations at a broker. The first one is $\overline{d_1} \wedge \overline{l'_1} \wedge \overline{l'_0}$. The destination restriction $\overline{d_1}$ means d_0 could be 0 or 1 but d_1 could just be 0. Hence the destination would be 01 or 00. Then $\overline{l'_1} \wedge \overline{l'_0}$ means that for any topic which destination is 01 or 00, the next location will be 00. The other situation, $d_1 \wedge l'_1 \wedge l'_0$ indicates that when the destination is 10 or 11, the next location will be 11. In our case a broker acts as a router and a policy is associate with each topic.

For a simple model with several components (e.g. routers, sensors or other end nodes), using less than 5 Boolean variables to represent the IP addresses is

feasible. However, in the real word, the IP addresses should be represented by 32 Boolean variables. If one designs a model using the IPv4 address structure, it will need up to 2^{32} different states, which may lead to a state explosion. In [12], the authors' propose a basic model that has five key identity variables; two of them (ports and port ids) are 16 bits long and the rest (IPsource, IPdestination and location) that are all 32 bits. In this model, there are thus 2^{128} possible states. In order to get rid of the state explosion problem, this paper proposes another way to build model. Since the publish/subscribe system is an overlay network, the number of brokers is much less than the number of routers in its under layer network. In our model we do not use 32-bit IP addresses. We use the natural numbers of IDs to handle the number of brokers. However, we potentially need policies for each topic and so the set of policies can be large.

Although model checking is a powerful technique, creation of appropriate finite state models for the systems being checked is still one of the stumbling blocks to using it. An important challenge of this research is how to build feasible models of publish/subscribe based MOM overlay networks to reduce the system to a finite state model, without eliminating the class of errors that the we wants to check..

IV. Modeling a Realistic Large Scale MOM System

The authors' previous work [22], describes a model for a publish/subscribe based MOM. A simplified model with only six brokers was used to perform the model checking. Firstly, the authors provided an illustrated example with six brokers where each of them has a unique 4-byte IP address. Each broker has a number of publishers and subscribers linked to it. In order to simplify the evaluation of this six-broker model, only one broker failure is considered at a time. The work in [22] requires a user to manually input a routing table for the overlay network and the 4-bit IP address structure was implemented. In this paper, a MOM system with 50 brokers is based on the system developed at the IBM T.J. Watson Research Center [23].

Each broker has a set of topics (that it is either publishing or subscribing to). With this number of brokers, manually generating the corresponding routing table is complex and error prone. Even if the rerouting is computed using a load balancing mechanism, additional constraints can require manual correction. Therefore, we developed a tool that represents the network of brokers in a MOM system and records the topics that each broker publishes or subscribes to. In the tool, the link information refers to the links between a broker to its neighboring brokers. The link information and the selection of a specific routing algorithm (e.g. Djikstra's Algorithm) are used to automatically generate the routing table of the MOM overlay. We also implemented a NuSMV code generator that automatically generates a full NuSMV MOM model with all the necessary specifications for the topic reachability verification.

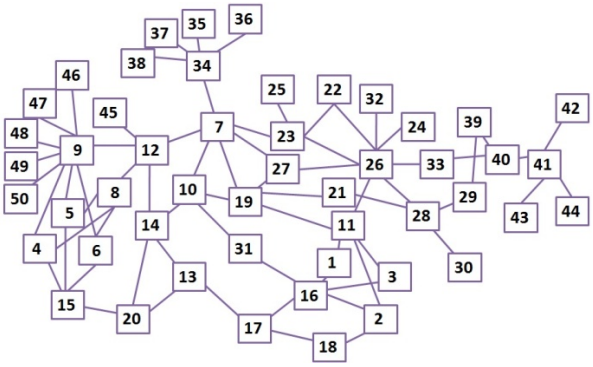

Figure 2: A Realistic Commercial-used MOM Overlay Network

Following parts introduce the whole processes of building and verifying a MOM system model in NuSMV.

A. Broker Information Collection

The first step of building this verification system is to collect the broker information and then setup the overlay network in NuSMV model checker. Thus the user needs to provide the total number of brokers in the overlay network (this will only need to be input once at the beginning) and the information for each broker. The information for each broker includes the ID of the broker that is used to identify the broker in the overlay network; the IDs of the neighboring brokers that have direct connections to this broker; and the published and subscribed topics that this broker dispatches. This information is used to further automatically generate a routing table that contains the shortest path between any two brokers in the network, and the subscribed and published topic distribution information. This information is stored into a hash map. Table I illustrates the information that is stored in the hash map structure for the first 3 brokers of Figure 2:

TABLE I. Structure of the stored MOM configuration

brokerID	neighbouring Brokers	publishedTopics	subscribedTopics
1	11,16	weather, films	music
2	11,18	sports	weather, stock
3	11,16	music	sports

1) Routing Table Generation

Here we define a path as a set containing all brokers that a message passes through from its source to destination during delivery. All possible paths for the overlay network is generated by using Djikstra's shortest path algorithm [26] based upon the link information, which refers to the broker and its direct connected brokers (neighboring brokers). The shortest paths for each source and destination are stored in a two dimensional array ($n \times n$), where n is the number of brokers in the network. This information can be represented as a matrix P where the rows correspond to the source broker IDs, the columns correspond to the destination broker IDs and the values are the corresponding sets containing the correspondent shortest paths:

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \cdots & \cdots & \cdots & \cdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,n} \end{bmatrix}$$

After generating the matrix P, the routing table for the overlay network is generated. The routing table can be represented as a matrix R where each row corresponds to a broker (Broker ID) of a topic and each column represents a destination broker (Broker ID). The value stored in the matrix shows the next hop according to its current location and destination.

$$R = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\ \cdots & \cdots & \cdots & \cdots \\ r_{n,1} & r_{n,2} & \cdots & r_{n,n} \end{bmatrix}$$

For example, for a particular topic the first 3 brokers of Figure 2, we could have the routing information shown in Table II:

TABLE II. EXAMPLE OF ROUTING TABLE

Loc.	Dest.	Loc.
1	3,7,11,19,20,21,22,23,24,25, 26,2728,29,30,31,32,33,34,35, 36,37,38,39,40,41,42,42,44	11
1	2,4,5,6,8,9,10,12,13,14,15,16, 17,18,20,31,45,46,47,48,49,50	16
2	7,8,9,11,12,19,21,22,23,24,25,26,27, 28,29,30,32,33,34,35,36,37,38,39, 40,41,42,43,44,45,46,47,48,49,50	11
2	1,3,10,14,16,31,	16
2	4,5,6,13,15,17,18,20,	18
3	3,7,11,19,20,21,22,23,24,25, 26,2728,29,30,31,32,33,34,35, 36,37,38,39,40,41,42,42,44	11
3	2,4,5,6,8,9,10,12,13,14,15,16, 17,18,20,31,45,46,47,48,49,50	16

In this table, the first column shows the current location of a message. The second column shows the identification of the destination broker. The last column provides the next hop of that message.

B. Generating the CTL Specifications in NuSMV for Every Stored Topic

We will build a NuSMV model of the delivering process for each topic. After having collected the necessary information for NuSMV to create the finite state machine for the MOM overlay network, the CTL specifications for checking the reachability of each topic needs to be generated for the input in the NuSMV model. What is verified here through CTL specifications is that each message from a certain publisher will reach all the required subscribers. The necessary information for building the model for each topic is the source broker ID, the destination broker ID and the broker ID of the current location. The NuSMV requires the following three state variables:

src is the source broker ID;

dest is the destination broker ID;

loc is the current location broker ID.

For example, if Broker 1 is the source of a topic and one of this topic's destination broker is Broker 13 The reachability verification can be represented in a CTL specification as:

$$(src = 1 \wedge dest = 13 \wedge loc = 1) \rightarrow AF(src = 1 \wedge dest = 13 \wedge loc = 13)) \quad (2)$$

The CTL specification (2) indicates that finally there will be a state where the current location of a topic coincides with its destination identification. This means that eventually the message will arrive at its destination.

Our implemented NuSMV model checker can test all the sources and destinations of a topic. If there are a large number of topics, the checker will take a long time to test all the sources and destinations entries. We require that for a pair of source-destination brokers, there is a path connecting them based on the routing protocol. However, this path could be part of another path or include other paths of other topics. We call a path of a topic that completely contains the paths of other topics as "super-path" and the contained paths are "sub-paths". For example, in Figure 2, a message that must go through a path from Broker 1 to Broker 13, it must go through Broker 16 and Broker 17. This path alone has 12 different sub-paths including itself (e.g. the number of possible combinations of all possible source and destination pairs). The path of a topic could be a sub-path of another already successfully checked path, consequently this sub-path need not be tested again.

With the increasing number of brokers, manually listing all possible paths (as presented in [22] for the six-broker model) is no longer feasible. An algorithm that can automatically generate all possible paths and find all sub-paths for a tested path is required for larger MOM systems. We integrated the detection of sub-paths and already processed paths in an algorithm that writes the CTL specification for each topic, significantly decreasing the overall testing time. In the algorithm, a matrix F is created to store the states indicating if a CTL specification has been already written for a specific path (source-destination pair) or not. In this matrix, if for a specific source–destination path a CTL specification has been written then the corresponding value is set to "true", otherwise the value is "false". Rows in matrix F indicate the source of a path, columns indicates the destinations of the path, and the corresponding boolean value indicates whether a verification check for that path has been written or not.

$$F = \begin{pmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,n} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,n} \\ \cdots & \cdots & \cdots & \cdots \\ f_{n,1} & f_{n,2} & \cdots & f_{n,n} \end{pmatrix}$$

The following pseudo-code describes the algorithm used for generating the NuSMV CTL specifications for every topic managed by the MOM:

```
1.  Initialize matrix F all to FALSE
2.  N:=total number of brokers
// For each broker, extract each subscribed topic
3.  For ID ← 1 to N
// Extract the subscribed topics of broker ID
4.      For t ← 1to Length Of (ID.subscribedTopics)
5.          TOPIC ← ID.subscribedTopic(t)
```

6. IDdest ← ID
// Going through all the brokers to extract the publishers of TOPIC
7. **For** IDSourcebroker ← 1 **to** N
// Search and extract the publishers of TOPIC
8. **For** n ← 1 **to** LengthOf (IDbroker.publishedTopics)
9. **if** TOPIC=IDbroker.publishedTopics(n)
10. **then** IDsource ← IDbroker
// Check matrix F to see if CTL formulas have already been written for this path and its sub-paths
11. **if** $f_{IDsource,\ IDdest}$ **!=** '**true**'
12. **then**
13. Write the corresponding CTL formula
// Search matrix P and extract the routing path of $p_{IDsource,IDdest}$
14. **For** $i \leftarrow 1$ **to** $l_p \leftarrow$ LengthOf ($p_{IDsource,IDdest}$)
15. **For** $j \leftarrow i+1$ **to** l_p
// Go through all the possible hops between IDsource and IDdest
16. Bsource ← $p_{IDsource,IDdest,i}$
17. Bdest ← $p_{IDsource,IDdest,j}$
18. $f_{Bsource,\ Bdest}$ ← "true"
19. $f_{Bdest,\ Bsource}$ ← "true"
20. **End** for j
21. **End** for i
22. **else if** goes to step 7
// The path and all sub-paths between IDsource and IDdest for TOPIC have been flagged as "true"
23. **End** for n
24. **End** for IDbroker
// All publishers of TOPIC have been searched and written into CTL formulas
25. **End** for t
// All the subscribed topics of Broker ID were processed
26. **End** for ID
// All the subscribed topics of all brokers were processed
// consequently, all CTL formulas for all sources and destinations of all topics have been processed

Table III displays a comparison between using and not using the sub-path detection algorithm for the fifty-broker model.

TABLE III. SUB-PATH DETECTION PERFORMANCE COMPARISON

Number of topics	Average number of paths to be tested		Average time to generate NuSMV code (ms) over 50 experiments on each different number of topics	
	Sub-path detection	No sub-path detection	Sub-path detection	No sub-path detection
10	30	50	2	7
50	120	250	3	27
100	200	500	7	67
500	468	2500	15	183
1000	550	5000	18	318
5000	570	25000	21	7942
10000	588	50000	37	58627

In order to assess the performance of our sub-path detection algorithm, we automatically generate different numbers of topics and assign each topic to 5 subscribers. From Table III, we can see that by using the sub-path detection the number of paths to be verified by NuSMV and the total time of generation are significantly reduced.

C. Loop Detection

Our implemented NuSMV model checker also provides a way of detecting the existence of loops in the routing table. The aim of loop detection is to find out whether a message goes back to a broker that it has already passed through. Formula (3) shows an example rule for detecting a loop path at Broker 1:

$$src = 1 \wedge EX(AF(src != 1)) \qquad (3)$$

Formula (3) will be true if a message does not pass through Broker 1 again after passing through it once. However, it is possible that a broker is the source broker as well as the destination broker at the same time (it has a subscriber of a specific topic, but at the same time, this broker also has a publisher to the same topic). Formula (3) will be evaluated to false as the source and destination broker is the same for that topic. There are two ways to overcome this problem. The first is to extract all the topics whose sources and destinations are in the same broker. The second way would be to use Timed CTL to impose a restriction in Formula (3) where it would only evaluate to false if and only if the message again passes through the same broker after it has traversed one or more hops (this is being added in our work).

V. THE MOM MODEL IN NUSMV

As mentioned in section II, the key task for constructing a NuSMV model is to find a suitable finite state model to replace the original system. We model the entire MOM overlay network as a finite state machine, while each state is defined by a different broker ID. The state transition of the overlay network is determined by the topics and the routing protocol in the overlay network. The routing protocol in this model is the shortest path. A topic starts from its source broker and follows its shorted path route until it reaches its destination broker.

Each of the three state variables for framing the NuSMV model (*src* , *dest* and *loc*) has the set of broker IDs (e.g. 1,2,…,49,50) as domain. So there are 1.25×10^5 possible states combinations for our model. All the variables are given initial value and rules for their next states transition. The initial values of *loc* is the initial value of *src* as a topic's initial current location is its source broker.

In the CTL specification the initial values for *src* and *dest* keep the same throughout. The rules for the next state transition of *loc* follow the matrix R (depending on the current value of *loc* and the value of *dest*) and are similar to formula (2) above.

For example, following Figure 2, if Broker 1 has a topic about 'weather' and Broker 13 subscribes to this

topic, then the, the state transition diagram is showing as the following figure:

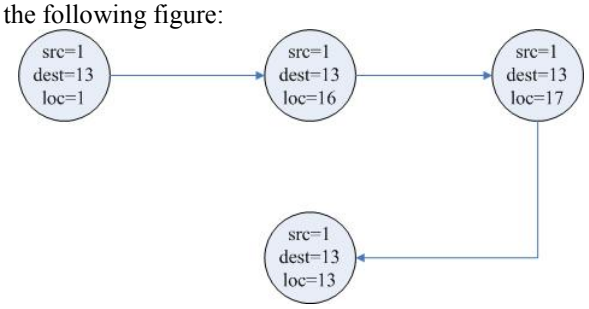

Figure 3: The state transition diagram segment (one source and one topic)

The first transition for this topic is from state $src=1, dest=13, loc=1$, and results in the next states $src=1, dest=13, loc=16$, and so on.

VI. Verifying Failures in a Large Scale MOM System

The two main types of failure within a MOM system are path failure or degradation beyond acceptable limits and broker failure. There are many reasons (e.g. path or buffer overload, power outages). that can lead to a failure. Quite a few researchers have devoted time to load balancing and provide measures to prevent failures. However, failures may still happen and a robust system needs to respond quickly to such incidents. In this research, since the user needs to manually input brokers' information, incorrect configuration may happen. It is possible that the user may forget a link or accidently isolate a broker. This paper proposes a way to verify failures on paths and on brokers.

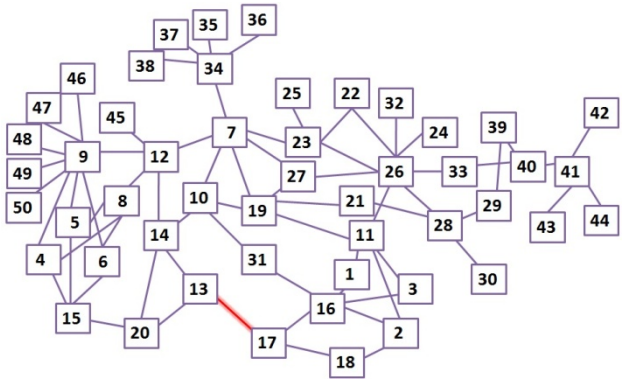

Figure 4: Failure of Direct Path between Broker 13 and 17

In previous sections, an integrated model for verifying the reachability for all topics has been presented. The model checker will terminate with either '***true***' or '***false***' to show the availability of a path or otherwise. All unavailable paths will be noted down and then their one-hop sub-paths will be tested again to locate the failed link(s). In Figure 4, let's assume that the direct path from Broker 13 to 17 has failed (we name this failed path as ***P1***). Since P1 failed, all paths that involve P1 will terminate with '***false***'. For example, the path from Broker 12 to 18, which including P1, will terminate with '***false***'. Then, its one-hop sub-paths which are the direct path from Broker 12 to 14, the direct path from Broker 14 to 13, the direct path from Broker 13 to Broker 17 and the direct path from Broker 17 to 18 will be retested. In this scenario, only the direct path from Broker 13 to 17 will terminate with '***false***' and then we can locate the failure path.

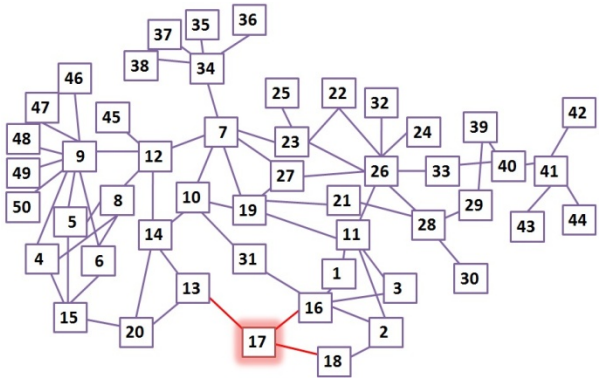

Figure 5: Failure of Broker 17

If a broker fails, all the direct paths traversing the failed broker will be unreachable. By the same measure, as shown in Figure 5, the direct path from Broker 13 to 17, the direct path from Broker 16 to 17 and the direct path from Broker 18 to 17 will test unreachable. There is then a high probability that Broker 17 has failed.

After locating the failure paths and brokers, the user can revise the topology and compensate for the failures.

VII. Conclusion and Future Work

Message oriented middleware has been widely used in cyber-physical systems, such as some wireless sensor networks [24] and Internet-enabled enterprise systems [25]. However, the primary disadvantage of many MOM systems is that they require an extra component in the architecture, the message transfer agent (i.e. the message broker). As with any system, adding another component can lead to a reduction in performance and reliability, and can also make the system as a whole more difficult and expensive to maintain. The goal to this research is to find a suitable means to retain the performance and reliability for a MOM system after a failure.

This work provides a code generator to automatically generate a MOM overlay checker when the links and topics of each broker are provided. Then the reachability of all topics are tested and the loops within the topology are detected. Also configuration could be manual (most would be automatic, but manual override may be done by the user) then our work is essential. Moreover, if some paths or brokers fail (could be the result of a misconfiguration), our checker quickly locates the failure and informs the user. A new configuration can be performed by the user based on the feedback provided by the model checker. The code generator that this research implements allows any user to build a model checker for a MOM based publish/subscribe system, even if the user has little knowledge of programming languages or of NuSMV.

As the number of brokers increases, manually generating a routing table for a large-scale MOM system becomes extremely complex. As mentioned before, this work designed a tool to assist users to build a model. By using the tool, users can simply input link information of the brokers, which is much easier to provide than a complete routing table, and the tool automatically generates a model from this link information. In this research, the routing table is generated based on the shortest path algorithm although it is possible to verify the contents of the table no matter how it is created (i.e. including static entries). In future work, we will provide more requisites for verifying a realistic MOM system. One of them is the ability to verify the reachabilty of topics when no predefined routing table is available. Moreover, since many MOM systems are expected to provide an efficient and high quality messaging service, another important requisite is to check if the rules for guaranteeing that end-to-end latency constraints are met by every broker. These new features are being added in our research work.

REFERENCES

[1] Xiao-FeiAn, Li-Ying Bian.'Design of Message-Oriented Middleware of Distance Teaching Platform Based on Distributed Message Control' 2010 International Conference on Computational Aspects of Social Neworks.

[2] Oracle Corporation, Java Message Service API Rev. 1.1, 2002. Available at http://java.sun.com/products/jms/.

[3] DDS: Data distribution service for real-time systems. http://www.omg.org/technology/documents/formal/data_distribution.htm

[4] E.Clarke, O. Grumberg, and D. Long. Model Checking 1990.

[5] M. C. Browne, E. M. Clarke and D. Dill. Automatic circuit verification using temporal logic: Two new examples. In Formal Aspects of VLSI Design. Elsevier Science Publishers (North Holland) 1986

[6] M. C. Browne, E. M. Clarke, D. Dill and B. Mishra. Automatic circuit verification using temporal logic. IEEE Transactions on Computers, C-35(12): 1035-1044. 1986

[7] M. C. Browne, E. M. Clarke and O. Grumberg. Characterizing finite kripke structures in propositional temporal logic. Theoretical Computer Science, July 1988

[8] Martin Kot. 'The State Explosion Problem' 2003[online]: http://www.cs.vsb.cz/kot/down/Texts/StateSpace.pdf

[9] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers, C-35(8), 1986

[10] M. Pistore and M. Roveri'The NuSMV Model Checker' 2003[online]: http://www.cse.iitd.ernet.in/~sak/courses/foav/nusvm-iitd-2.pdf

[11] HabtamuAbie. 'Adaptive Security and Trust Management for Autonomic Message-Oriented Middleware'MobileAdhoc and Sensor Systems, 2009. MASS'09. 2009

[12] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. ElBadawi. 'Network Configuration in A Box: Towards End-to-End Verification of Network Reachability and Security' In Proceedings of the 17th IEEE International Conference on Network Protocols (ICNP), pages 123–132, 2009.

[13] Richard Alimi, Ye Wang, and Y. Richard Yang.'Shadow configurationas a network management primitive' In SIGCOMM ' 08: Proceedings ofthe ACM SIGCOMM 2008 conference on Data communication, pages111–122, New York, NY, USA, 2008. ACM.

[14] G. Abowd, R. Allen, and D. Garlan. Using style to understand descriptions of software architecture. In Proceedings of SIGSOFT'93: Foundations of Software Engineering, Software Engineering Notes 18(5). ACM Press, December 1993.

[15] D. J. Barrett, L. A. Clarke, P. L. Tarr, and A. E. Wise. A framework for event-based software integration. ACM Transactions on Software Engineering and Methodology, 5(4):378–421, October 1996.

[16] J. Dingel, D. Garlan, S. Jha, and D. Notkin. Reasoning about Implicit Invocation. In Proceedings of the Sixth International Symposium on the Foundations of Software Engineering (FSE-6), Lake Buena Vista, Florida, November 1998. ACM.

[17] J. Dingel, D. Garlan, S. Jha, and D. Notkin. Towards a formal treatment of implicit invocation. Formal Aspects of Computing, 10:193–213, 1998.

[18] D. Garlan and D. Notkin. Formalizing design spaces: Implicit invocation mechanisms. In VDM'91: Formal Software Development Methods, pages 31–44, Noordwijkerhout, The Netherlands, October 1991. Springer-Verlag, LNCS 551.

[19] Garlan D, Khersonsky S, Kim J S. Model checking publish-subscribe systems[M]//Model Checking Software. Springer Berlin Heidelberg, 2003: 166-180.

[20] H. Zhang, J. S. Bradbury, J. R. Cordy, and J. Dingel. Implementation and verification of implicit-invocation systems using source transformation. In Proc. of the 5th Int. Wkshp. on Source Code Analysis and Manipulation (SCAM05), pages 87–96, 2005.

[21] Luciano Baresi, Carlo Ghezzi, and Luca Mottola, Piazza Leonardo da Vinci, Milano.*'On Accurate Automatic Verification of Publish-Subscribe Architectures'*2007

[22] Y. Jia, E. Bodanese, J. Bigham, 'Checking the Robustness of a Publish/Subscribe Based Message Oriented System', *IV International Congress on Ultra Modern Telecommunications and Control Systems*, pp 291-296, October 2012.

[23] Hao Yang, Minkyong Kim, KyriakosKarenos, Fan Ye, and Hui Lei,'*Message-Oriented Middleware with QoS Awareness'*IBM T. J. Watson Research Centre 2009.

[24] Souto E, Guimarães G, Vasconcelos G, et al. *A message-oriented middleware for sensor networks[C]*//Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing. ACM, 2004: 127-134.

[25] Tran P, Greenfield P, Gorton I. *Behavior and performance of message-oriented middleware systems[C]*//Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on. IEEE, 2002: 645-650.

[26] Shaikh-Husin N, Hani M K, Seng T G. *Implementation of recurrent neural network algorithm for shortest path calculation in network routing[C]*//Parallel Architectures, Algorithms and Networks, 2002. I-SPAN'02. Proceedings. International Symposium on. IEEE, 2002: 313-317.