Verification of P2P Live Streaming Systems Using Symmetry-based Semiautomatic Abstractions

Pedro de Carvalho Gomes
KTH Royal Institute of Technology
Stockholm, Sweden
pedrodcg@csc.kth.se

Sergio Vale Aguiar Campos
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
scampos@dcc.ufmg.br

Alex Borges Vieira
Universidade Federal de Juiz de Fora
Juiz de Fora, Brazil
alex.borges@ufjf.edu.br

***Abstract*—P2P systems are one of the most efficient data transport technologies in use today. Particularly, P2P live streaming systems have been growing in popularity recently. However, analyzing such systems is difficult. Developers are not able to realize a complete test due the due to system size and complex dynamic behavior. This may lead us to develop protocols with errors, unfair or even with low performance. One way of performing such an analysis is using formal methods. Model Checking is one such method that can be used for the formal verification of P2P systems. However it suffers from the combinatory explosion of states. The problem can be minimized with techniques such as abstraction and symmetry reduction. This work combines both techniques to produce reduced models that can be verified in feasible time. We present a methodology to generate abstract models of reactive systems semi-automatically, based on the model's symmetry. It defines modeling premises to make the abstraction procedure semiautomatic, i.e., without modification of the model. Moreover, it presents abstraction patterns based on the system symmetry and shows which properties are consistent with each pattern. The reductions obtained by the methodology were significant. In our test case of a P2P network, it has enabled the verification of liveness properties over the abstract models which did not finish with the original model after more than two weeks of intensive computation. Our results indicate that the use of model checking for the verification of P2P systems is feasible, and that our modeling methodology can increase the efficiency of the verification algorithms enough to enable the analysis of real complex P2P live streaming systems.**

I. Introduction

P2P systems are extremely efficient in disseminating data to a large number of users. In particular, P2P live streaming systems can be used to broadcast video to very large numbers of viewers without overloading centralized servers [19]. However, making sure that these systems behave as expected can be difficult, due to their complex dynamic behavior.

Some of the most commonly used methods to check a P2P live streaming behavior are simulation and data collection from real transmissions. Both methods are very useful, but do not solve all problems. Simulation is very efficient, but due to its nature, can miss important but uncommon events, and as a consequence, its results are not guaranteed [17]. The same is true for analyzing real transmissions, which have the additional disadvantage of being complex to set up and run.

In this work, we propose the use of the formal method Model Checking to check P2P live streaming systems behavior and its correctness. We also provide an exhaustive analysis of a P2P system model using our approach. We focus on the *liveness* properties, which are the ones that guarantee that some condition may happen. We use, for this purpose, the existential subset of the *Computation Tree Logic* (CTL), named $\exists CTL$.

Despite the success of Model Checking on the verification of a large number of computation systems, it is susceptible to the state-space explosion problem. This problem is caused by the fact that the number of states in a model is exponential to the number of variables it contains [17]. This severely limits the complexity of models that can be verified.

The use of abstractions is one of the main techniques to mitigate the state-space explosion. It generates smaller abstract models by removing or grouping states and transitions from the original model [1]. Another technique to produce compact models is symmetry reduction [9]. It is applicable on models that present replicated structures. Its key idea is that some states present symmetrical behavior for the verification of some properties. So, instead of exploring all states, the exploration of only one of these symmetric states is sufficient.

Although such techniques have been widely studied, and even combined, they still have limitations. The *symmetry reduction* has a high computing cost to determine the symmetry between states [8]. The *abstraction reduction* can lead to many inconclusive results if performed naively.

This work aims to address such limitations and assist the model checking task by using the intuitive knowledge which the software engineer has about the model. We present a methodology to assist the person who will check the model to identify the symmetry in a formal and simple way.

The main contribution of this work is a semiautomatic technique to remove components from models of reactive systems. This new methodology removes parts (components) of the model by human intervention, but without altering its description; we do it by providing logical constraints that eliminate transitions, thus reducing the reachable states. The decision about which components of the model to eliminate is made using the identified symmetry.

We validate the methodology on the model of GridMedia [19], a well known P2P live streaming system. Although, we may also use our new methodology in others P2P system as file sharing and tree-based live streaming applications. The quantitative results show that the abstractions used in this work

have achieved considerable reductions for both verification time and size of the model. In fact, it was not possible to complete the computation of the number of reachable states for the original GridMedia model after two weeks. The same computation finished in less than 3 minutes using the new methodology. Moreover, the maximum number of reachable states found was less than 10^6, many orders of magnitude lower than the approximately 10^{22} possible states, which demonstrate the power of model reduction through elimination of components.

The reminder of this paper is organized as follows: we present related work on Section II. The section III discusses the theoretical foundations for the performed reductions. We present our new reduction methodology on section IV. On section V we present the GridMedia model, and quantitative results. Finally, section VI concludes our work.

II. Related Work

The use of abstractions in Model Checking has a rich literature. Over-approximation are used in [1] to verify properties in $\forall CTL$ logic. A dual result is presented in [2] for the verification of abstract models by under-approximation, using $\exists CTL$. We based much of our work on [3], which describes the verification of liveness properties for reactive systems. [4] presents a technique for the automatic abstractions and refinements when verifying $\forall CTL$ properties. More recently [5] adapted the consolidated techniques of abstractions to Directed Model Checking. Our approach differs for the cited ones in the way we implement such abstractions, by removing components using direct specifications.

The exploitation of the symmetry in models was proposed at the same time by [6] and [7]. Both use the concept of permutations over the states of a model to determine its symmetry. We borrow the theoretical definitions from [9], which are also used in [10]. The present work differs from the latter in some ways: first, we define the symmetry over components, not over states. The number of components is severely smaller than the number of states. Thus, the computation of symmetry is significantly easier. Moreover, we don't perform symmetry reduction; we use symmetry only as the orientation to perform the abstractions.

Previous works combine symmetry and abstractions. In [11] over-approximations and symmetry are combined to check $\forall CTL$ formulas. Our work checks properties in $\exists CTL$ Under-approximation and symmetry are combined in [13], focusing on the falsification of properties. The technique is applied to On-the-fly Model Checking, which constructs the model at the same time it verifies the property. This work differs from ours mainly because we construct the abstract model before checking.

Although there are several works about formal verification of network protocols, there are few that focus on P2P networks and/or use Model Checking. Among these, [14] and [15] verify distributed hash table protocols; the first uses automated theorem proving, and the latter uses Model Checking. Moreover, Probabilistic Model Checking was used in [16] to verify a reorganizing protocol for P2P networks. To the best of our knowledge, the present work is the first one to verify a P2P Live Streaming system.

III. Preliminaries

The following sections present the theoretical foundation for the performed reductions. Some explanations omit formal details, to help the intuitive comprehension.

A. Under-approximations

The abstraction by under-approximation systematically eliminates states and transitions. It creates abstract models that, although have smaller number of behaviors, do not admit behaviors that are nonexistent in the original model. Let $M = (S, S_0, R, L)$ be the standard definition of a Kripke structure, where S is the set of states, $S_0 \subseteq S$ is the set of initial states, $R : S \times S$ the transition relation, and $L : S \to \mathcal{P}^{AP}$ is the mapping from a state to its set of atomic propositions. An abstract Kripke structure is defined as $\widehat{M} = (\widehat{S}, \widehat{S_0}, \widehat{R}, \widehat{L})$, where:

- $\widehat{S} \subseteq S$.
- $\widehat{S_0} = \widehat{S} \cap S_0$.
- $\widehat{R} \subset R$, where $(s, s') \in \widehat{R} \Rightarrow s, s' \in \widehat{S}$.
- $\widehat{L} = L$.

By the definition of $\widehat{R}$, it is trivial to notice that not all paths in the original model also exist in the abstract model. Conversely, if a path exists in an abstract model it necessarily also exists in the original model. Notice that the definition above also holds for the case where only transitions are eliminated. The use of $\exists CTL$[2], the fragment of CTL with existential operators only, guarantees the correctness of model checking with abstractions in this scenario. Intuitively, if the verification of a $\exists CTL$ formula ϕ is correct w.r.t the abstract model, it implies that the verification found at least one path that satisfies ϕ; thus ϕ is also correct w.r.t to the original model since all the paths in an abstract model also exist in the original.

The check of $\exists CTL$ formulas over models with under-approximations for the purpose of falsification, i.e., proves that an undesirable condition may happen, is also valid.

B. Permutation Groups

A permutation σ over a set A is a bijective function defined as $\sigma : A \to A$. The identity permutation e is the one that maps each element of the set A into itself. A permutation like $a_1 \mapsto a_2, a_2 \mapsto a_3, ..., a_{k-1} \mapsto a_k, a_k \mapsto a_1$ is called a cycle and is written as $(a_1 a_2 a_3 ... a_k)$. Any permutation can be written as the functional composition of disjoint cycles. The usual notation in works about symmetry in Model Checking is to represent the functional composition similarly to concatenation. Therefore, $f(g(x)) = f \circ g = fg$.

A permutation group G is a set of permutations associated with the operation of functional composition, such that:

- The identity permutation $e \in G$.
- For every permutation $\sigma \in G$ exists a permutation $\sigma^{-1} \in G$ such that $\sigma\sigma^{-1} = e$.

- For all permutations $\sigma_1, \sigma_2 \in G$, $\sigma_1\sigma_2$ is also in G.

Permutations $\sigma_1, \sigma_2, \sigma_3, ..., \sigma_k$ are said to be the generators of a permutation group G, if G is the closure of the set $\{\sigma_1, \sigma_2, \sigma_3, ..., \sigma_k\}$ under the operation of functional composition. Formally, $G = \langle(\sigma_1, \sigma_2, \sigma_3, ..., \sigma_k)\rangle$.

In this paper we use permutations over the finite set of states S of the Kripke structure M that represents the system. However a permutation can also be stated in terms of the components of the model. Let p_i and p_j be two identical components of a system. The function $\sigma(p_i) = p_j$ indicates the permutation of all the states where the variables of p_i and p_j have the same valuations.

C. Symmetry Groups

A permutation group G is also a symmetry group with respect to a Kripke structure M if all its permutations preserve the transition relation and the initial states of M. $\forall s, s' \in S, \forall \sigma \in G[((s, s') \in R \iff (\sigma(s), \sigma(s')) \in R) \wedge (s \in S_0 \iff \sigma(s) \in S_0)]$.

Given a symmetry group G, we can partition the set of states S into equivalence classes called orbits. Formally, the orbit of a state s is a set of states $\theta(s) = \{t | \exists \sigma \in G$, where $\sigma(s) = t\}$. Thus, we can generate a reduced model from the original one by selecting one representative from each orbit θ, which we call $rep(\theta(s))$. This reduced model is known as abstract model.

Formally, the abstract model of a Kripke structure M, with respect to a symmetry group G is defined as $M_G = (S_G, S_G^0, R_G, L_G)$, where:

- $S_G = \{\theta(s) | s \in S\}$ is the set of orbits of the state in S.
- $S_G^0 = \{\theta(s) | s \in S_0 \wedge s = rep(\theta(s))\}$.
- $R_G = \{(\theta(s), \theta(s')) | (s, s') \in R\}$.
- $L_G(\theta(s)) = L(rep(\theta(s))$.

The fact that G is a symmetry group makes all initial states $s \in S_0$ at M have a representative of its orbit also in S_G^0. For the same reason R_G is well-defined and independent of the chosen representatives for each orbit.

IV. Methodology

In this section we present the premises for the application of our reduction methodology. Moreover we present an overview of how to perform the formal symmetry identification and components abstraction. The definitions presented here are illustrated in Section V.

A. Modeling

The application of our methodology relies on the presence of some modeling patterns, such as the separation between internal behavior of a component and its interface with other components. In this section we present how each of these patterns is important to either implement the semiautomatic abstractions or to formally identify the symmetry. Fortunately these patterns are common for the model of many reactive systems. For these systems, the concept of a component means an independent logical entity that communicates with other entities, and acts based on its interactions. In a typical symmetrical model we have components of the same type. This is the case of the P2P Live Streaming network presented in Section V, where components are the participants of the transmission.

1) Internal Representation: The internal representation refers to the description of the finite state machines of the components in the model. It contains the declarations of the state variables and the transitions rules of its values based on the current and next states.

Its definition shall implement the concept of semiautomatic abstractions because the removal of a component from the model requires that the transitions that modify their values are disabled. In other words, a component which never changes the values of its state variables is considered to be removed from the model. Section IV-C shows the assumptions that the internal representation must meet to enable the semiautomatic abstractions.

2) Communication Interfaces: A communication interface defines how the components interact. The input of a logic signal in a circuit or a communication message on a network illustrates this concept. They are a common medium between the internal and external representations. Typically they are implemented as a shared state variable between the communicating components.

3) External Representation: The external representation refers to the declaration of the components of the model and how they are organized. Some examples are the definitions of the connections of logical components in a circuit or the topology of a network. The analysis of the external representation is what determines the symmetry between the components. Moreover, the semiautomatic abstractions alter the external representation by logically eliminating connections between components, or removing an entire component by eliminating all its communication.

B. Generation of the symmetry group

In this work, the symmetry guides which parts should be removed. Its use is justified by the smaller loss of behaviors it causes [8]. Intuitively, if the property in analysis preserves the symmetry of the model, we can consider only one of the symmetrical components. This representative contains the necessary information for the verification. We can also illustrate this concept in terms of variables and states. If the property preserves the symmetry of the model, then the valuations for the state variables of the representative component represent the same valuations of variables from the removed symmetric components.

The definition of symmetry between the components is made by generating the symmetry groups. As mentioned in Section III-B, these groups are sets of permutations that preserve the transition relation. The problem is the same as the classical problem of isomorphism in two graphs [17], and should be repeated for all possible permutations. Our method is efficient because it performs such computation in the component-level, in contrast to state-level. As a consequence, it manipulates a significantly smaller number of elements.

The model's external representation is analyzed to determine the symmetry group. If the system has been modeled as described in Section IV-A then the task becomes easier because the model can be viewed as a graph where the components are the vertex, and connections between components are the edges. If the permutation of two components maintains the transition relation in the graph of the external representation, then this permutation will be part of the symmetry group. Recall that a permutation between two components means the permutation among the states that have symmetric valuations for the state variables of both, as defined in Section III-B.

C. Semiautomatic abstractions

Semiautomatic abstractions are implemented in the level of components, by human intervention, but without modifying the model's description. This is accomplished by using direct specifications, which are semantic constructions of modeling languages (e.g. SMV[18]). Direct specifications are stated as propositional formulas, and the valid transitions are the ones that evaluate to true for such formulas. We use direct specifications to limit the transitions of the state variables of a component to only the transitions that do not alter the variables. The fact that the values of the state variables will not be altered implies that logically the component is no longer part of the model. Consequently the number of reachable states is reduced since the state variables of the removed component do not contribute to the number of possible valuations in the model.

The methodology defines that the internal representation of the components should provide a default value for the communication variable. The key idea is that if there is no communication, then the variables of a component would never change its values. In other words, the state of a component would never change. This is a valid and fairly common situation for many reactive systems, where components remain idle (keep current state) until receiving data from another component. Conversely, if there is no communication, the component will always keep the same values for the state variables and hence the states defined by other valuations become unreachable.

The semiautomatic abstractions eliminate states and transitions that impact the verification of the model. To illustrate, we can think of a model with two components, each with a single variable ranging from one to three. If we force one of the components to always keep its variable to zero, the valuations of the system where one component's variable is two, and another component's variable is three would not be represented.

So, the abstractions presented in this work are under-approximations. Consequently, the methodology of this study uses only formulas in the $\exists CTL$ logic. As we mentioned in Section III-A, if the verification of a property holds for the abstract model, it also holds for the original model. But if does not hold, the result is inconclusive. In this case the abstractions should be removed one by one, and the check should be performed again until one of the two conditions hold: the property is proved to be true, which is a valid result; or there are no more abstractions to be removed and the property will be checked over the original model, which may be unfeasible.

Finally, only formulas that do not reference the components being removed are valid. Although the semiautomatic abstractions logically remove a component, in fact it is still present in the model. So, if the $\exists CTL$ formula being verified has any reference to variables of the removed component, the model checker will judge it as valid. Consequently it may yield incorrect results because it contains reference to a component that will never change the values of its variables.

V. The GridMedia Model

Recently the P2P architecture began to be used by applications broadcasting live content. Such applications are called P2P Live Streaming. They inherit all the characteristics of P2P systems for file sharing, in addition to a strong constraint of low-latency on the transmission. Most of these systems implement an algorithm based on the BitTorrent network to distribute data. On it, the original information is fragmented into several pieces and distributed among the participants of the network; then the participants try to rebuild the original information by exchanging data with its partners. In this work we model[1] a subset of the GridMedia network [19].

In the GridMedia, information is generated solely by a special node, called server. Unlike the concept of same name in the client-server model, this node does not serve all participants. It just generates the live transmission data, divides into small pieces called chunks, and distributes each chunk to few participants. Each participant tries to reconstruct the original information by requesting to its neighbors the missing chunks, and forwards to them the chunks it has, but they have not.

The delivery of the information takes place after three steps, and is known as *pull* method. First, each node informs its partners constantly which chunks it has and which chunks it needs through a data structure called chunk map. After receiving the partner's chunk map, the node asks for a missing chunk. The partner receives the request and then replies sending the requested chunk. GridMedia has a complementary method to pull, called push, which was not considered in our model. Figure 1 illustrates the pull scheme.

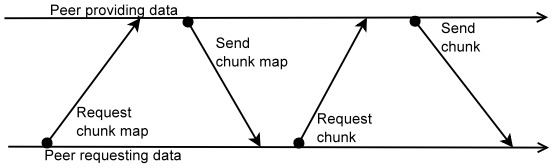

Fig. 1. Data delivery by the pull method

A. Internal Representation and Interfaces

The internal representation refers to the description of the state machines of the network participants. The model of the GridMedia network contains two types of participants: the

[1] Available at http://www.csc.kth.se/~pedrodcg/files/gridmedia.smv

server, that generates the content to be distributed, splits into chunks; and nodes, which receive and forward chunks to their neighbors to reconstruct the original information. The model defines six types of messages plus the special value `null`, which indicates that no message was received.

B. External Representation

The topology of the GridMedia network implemented in this work is shown in figure 2. It has the *server* connected to nodes *A* and *B* in a first level. In a second level there are the nodes *C* and *D*, which have connections to both *A* and *B*, and between themselves.

Note that many relevant interactions are represented: the direct communication between the server and clients is represented by $(server \rightarrow A)$ and $(server \rightarrow B)$; the communication between clients on the first level with clients on the second level happens in $(A \leftrightarrow C)$,$(A \leftrightarrow D)$,$(B \leftrightarrow C)$ e $(B \leftrightarrow D)$; finally, the connection between two clients at the same level is represented by $(C \leftrightarrow D)$. Note that there are no incoming edges to the server; this is because the server only sends data, and does not request anything. The modeling of the topology was made in the SMV language and it contains the asynchronous composition of five processes.

Despite this simple example, it is easy to see that this topology is highly symmetrical. As the P2P network grows, we notice more and more groups similar to 2 appearing. Even with a large number of clients, there still symmetric parts in the P2P network which allows applying our technique.

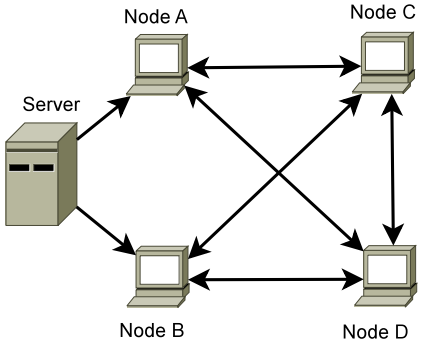

Fig. 2. Topology of the GridMedia network

C. Symmetry Group

The diagram representing the GridMedia network contains five components, being four of them identical. They are connected by twelve edges only. Thus, to generate the symmetry group we simply exchange the symmetrical components and verify if the transition relation is preserved. Table I shows the computation of the symmetry group.

Some permutations of identical components do not preserve the transition relation. I.e., not all edges in these permutations are also present in the original model. This was the case of $(Node_A Node_C)$ of $(Node_A Node_B Node_C Node_D)$. The strike-out edges do not exist in the transition relation of the original model. The permutations $(Node_A Node_D)$, $(Node_B Node_C)$ and $(Node_B Node_D)$ do not preserve the transition relation by analogy to $(Node_A Node_C)$, and were omitted.

TABLE I
GENERATION OF THE SYMMETRY GROUP

Edge	Permutation			
	$(Node_A\ Node_B)$	$(Node_C\ Node_D)$	$(Node_A\ Node_C)$	$(Node_A Node_B\ Node_C Node_D)$
$S \rightarrow A$	$S \rightarrow B$	$S \rightarrow A$	~~$S \rightarrow C$~~	$S \rightarrow B$
$S \rightarrow B$	$S \rightarrow A$	$S \rightarrow B$	$S \rightarrow B$	~~$S \rightarrow C$~~
$A \rightarrow C$	$B \rightarrow C$	$A \rightarrow D$	$C \rightarrow A$	$B \rightarrow D$
$C \rightarrow A$	$C \rightarrow B$	$D \rightarrow A$	$A \rightarrow C$	$D \rightarrow B$
$A \rightarrow D$	$B \rightarrow D$	$A \rightarrow C$	$C \rightarrow D$	~~$B \rightarrow A$~~
$D \rightarrow A$	$D \rightarrow B$	$C \rightarrow A$	$D \rightarrow C$	~~$A \rightarrow B$~~
$B \rightarrow C$	$A \rightarrow C$	$B \rightarrow D$	~~$B \rightarrow A$~~	$C \rightarrow D$
$C \rightarrow B$	$C \rightarrow A$	$D \rightarrow B$	~~$A \rightarrow B$~~	$D \rightarrow C$
$B \rightarrow D$	$A \rightarrow D$	$B \rightarrow C$	$B \rightarrow D$	$C \rightarrow A$
$D \rightarrow B$	$D \rightarrow A$	$C \rightarrow B$	$D \rightarrow B$	$A \rightarrow C$
$C \rightarrow D$	$C \rightarrow D$	$D \rightarrow C$	$A \rightarrow D$	$D \rightarrow A$
$D \rightarrow C$	$D \rightarrow C$	$C \rightarrow D$	$D \rightarrow A$	$A \rightarrow D$
Preserve the transition relation	Yes	Yes	No	No

The permutations between nodes A and B, and between the nodes C and D preserve the transition relation. Thus we conclude that the permutations $\langle (Node_A Node_B), (Node_C NodeD) \rangle$ are generators of the symmetry group $G = \{e, (Node_A Node_B), (Node_C Node_D), (Node_A Node_B)\ \ (Node_C Node_D)\}$ with respect to the original model.

D. Introduction of Abstractions

The abstractions in this work refer to the removal of entire components from the model and the consequent removal of states. The removal of a component may be the elimination of a connection between two participants, represented by the pair of variables that simulate message boxes, or the removal of a participant in the network, represented by the removal of all the connections that it maintains. The connection between two participants is abstracted by inputting the `TRANS` statements that restrict the valid transitions to those where the values of the communication variables is unchanged. The `TRANS` statement below removes the connection between node *A* and node *C*. The abstraction of a participant is made similarly, by inputting direct specifications that restrict any changes to all its communication variables.

```
TRANS (next(node_A.connection2.event)=null)
    & (next(node_C.connection1.event)=null)
```

E. Abstract Models

In this section we construct the abstract models based on the symmetry group, and on the property being checked. In the properties, nodes are represented by a process named `Node_{A,B,C,D}`. Each node has a variable named `chunk_buffer`. This is an array of two bits, where each bit represents one chunk of data: if set to `0`, the node does not have that chunk; `1` otherwise.

1) Property ϕ_1: Partial Data Reconstruction: The first property checks the existence of the scenario where one client that is in distance of size two to the server can recreate the original information before another node, at the same distance, receives any part of the information. The formula ϕ_1 below verifies this case, represented by the chunk buffer of $Node_D$ containing the two bits of data, and the buffer of $Node_C$ being empty:

```
φ1 =  E [  node_C.chunk_buffer=0B2_00
        U  node_D.chunk_buffer=0B2_11]
```

As described in Section IV-C, only components that are not referenced in ϕ_1 can be abstracted. Thus, the orbit generated by the permutation $(Node_C Node_D)$ cannot be used, since both of its nodes are part of the formula. The permutation $(Node_A Node_B)$ is the only one valid for generating the abstract model, and any of the nodes can be the representative of the orbit. We chose $Node_A$; consequently $Node_B$ is abstracted. The abstract model is represented in figure 3.

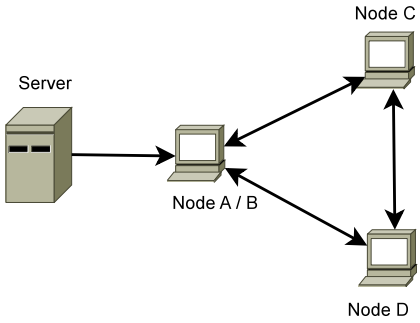

Fig. 3. Abstract model for ϕ_1

The verification of ϕ_1 holds for the abstract model. Thus it is also correct over the original model.

2) Property ϕ_2: Multiple Paths: An expected property in P2P networks is the possibility of nodes that are not directly connected to the server to receive all data before the clients connected to the server. This should be possible since the server partitions the original information and distributes different parts to the nodes connected directly to it. So it is expected that there is more than one path that information can take to reach any participant in the network. The formula ϕ_2 below checks the case where the nodes connected to the server have distinct data parts, but a node not connected to the server already has the full data:

```
φ2 =  EF  (node_A.chunk_buffer=0B2_10)
        &  (node_B.chunk_buffer=0B2_01)
        &  (node_C.chunk_buffer=0B2_11)
```

The two elements of permutation $(Node_A Node_B)$ are in ϕ_2 and cannot be used to abstract components. The only element of the permutation $(Node_C Node_D)$ mentioned in the formula is $Node_C$. Consequently we chose it as the representative from the orbit, and abstract $Node_D$. The abstract model is shown in figure 4.

The verification showed that ϕ_2 holds for the abstract model, and we can conclude that it also holds for the original model.

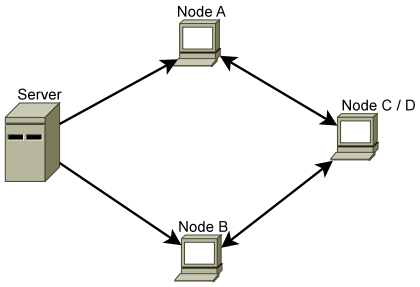

Fig. 4. Abstract model for ϕ_2

3) Property ϕ_3: Transmission Error: As mentioned in Section III-A, the check of $\exists CTL$ properties is also valid for falsification, i.e. prove that a bad condition may happen. This is the case for the property ϕ_3 below. It checks if there exists a situation where two nodes in different distances to the server, e.g. $Node_B$ and $Node_D$, may not receive any data:

```
φ3 =  EG (  node_B.chunk_buffer=0B2_00
          &  node_D.chunk_buffer=0B2_00)
```

The permutations $(Node_A Node_B)$ and $(Node_C Node_D)$ may be used to generate the abstract model, as both have members that are not in ϕ_3. In this case, $Node_B$ and $Node_D$ are automatically selected as representatives of the orbits, and $Node_A$and $Node_C$ are abstracted. The abstract model is presented in figure 5.

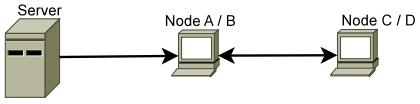

Fig. 5. Abstract model for ϕ_3

The temporal formula ϕ_3 holds for the abstract model. Thus we conclude that transmission can fail, and it is possible that no data is received at all.

F. Comparison of the reductions

The original model has a total number of states of approximately 10^{22}, referring to all the possible valuations over the state variables. Its complexity did not allow the calculation of the graph diameter (maximum shortest path between any two pairs of vertices), nor the number of reachable states after more than two weeks of intensive computation on a dedicated processing server.

TABLE II
COMPARISON BETWEEN THE ABSTRACT MODELS

Abstract Model	Diameter	Reachable States	Total Time (s)	CPU Time (s)
Property ϕ_1	41	500079	37,267	0,048
Property ϕ_2	41	814135	143,137	0,100
Property ϕ_3	28	793	0,201	0,016

By contrast the computations performed for all abstract models finished in less than three minutes in all cases. Table II shows a quantitative comparison between the verification of

the abstract models. It presents the diameter of the obtained graph, the number of reachable states, the CPU time spent to verify the property and the total time between the beginning and the end of the model checker process.

The most remarkable fact was the reduction of exponential order in the number of reachable states, from about 10^6 in the case of the abstract model for ϕ_2 to approximately 10^3 in the abstract model for ϕ_3. It shows the magnitude of the reduction in a model when a single component is abstracted. Similar result can be observed by comparing the diameters. In the abstract models for ϕ_1 and ϕ_2, where only one component was abstracted, the diameter size was 41; in the model for ϕ_3, which had two components removed, the value dropped to 28.

VI. Conclusion

The methodology presented in this paper makes it possible to effectively use formal methods, in particular model checking in the analysis of large realistic P2P systems. It reduces models in terms of components, in contrast to reduction by states. The reasoning about components is more intuitive since the modeling is done at this level, and is more practical because it deals with a smaller amount of objects. Consequently the identification of symmetry in the model is simpler.

Furthermore, the quantitative results show that the removal of components can result in significant reduction in the number of reachable states and can make possible the verification of models that were previously intractable. This was the case in our example, where the verification of properties in the original model did not finish after two weeks of intensive computation on a dedicated server. The same computations have finished in less than three minutes for all the abstract models.

The identification of the symmetry proved to be a strong evidence of how to choose the components to be abstracted. It has eliminated redundant behaviors and has minimized the information loss. This can be noticed by the fact that it was not necessary to refine the abstract model because of an inconclusive result.

One limitation of the technique is that it applies only to highly symmetric models. Fortunately this is the case for several real models, like the example of the GridMedia network. A second limitation is that the abstractions remove information which makes it only possible to check properties in $\exists CTL$.

Future work includes applying the methodology we propose to other kinds of P2P system as file sharing and to social networks. Moreover, we intend to extend our methodology to capture heterogeneous characteristics as processing capacity and upload bandwidth. Finally, we intend to evaluate our approach in a dynamic scenario, following real world P2P applications traces.

Acknowledgment

The authors thank Dilian Gurov for the valuable feedback.

References

[1] E. M. Clarke, O. Grumberg, and D. E. Long, "Model checking and abstraction," ACM Transactions on Programming Languages and Systems, vol. 16, no. 5, pp. 1512–1542, September 1994.

[2] W. Lee, A. Pardo, J.-Y. Jang, G. Hachtel, and F. Somenzi, "Tearing based automatic abstraction for ctl model checking," in ICCAD '96: Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design. Washington, DC, USA: IEEE Computer Society, 1996, pp. 76–81.

[3] D. Dams, R. Gerth, and O. Grumberg, "Abstract interpretation of reactive systems," ACM Trans. Program. Lang. Syst., vol. 19, pp. 253–291, March 1997.

[4] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement for symbolic model checking," J. ACM, vol. 50, no. 5, pp. 752–794, 2003.

[5] K. Drager, B. Finkbeiner, and A. Podelski, "Directed model checking with distance-preserving abstractions," Int. J. Softw. Tools Technol. Transf., vol. 11, no. 1, pp. 27–37, 2009.

[6] C. N. Ip and D. L. Dill, "Better verification through symmetry," in CHDL '93: Proceedings of the 11th IFIP WG10.2 International Conference sponsored by IFIP WG10.2 and in cooperation with IEEE COMPSOC on Computer Hardware Description Languages and their Applications. Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., 1993, pp. 97–111.

[7] E. A. Emerson, A. P. Sistla, and H. Weyl, "Symmetry and model checking," 5th International Conference on Computer-Aided Verification, vol. 697, 1993.

[8] S. Jha, "Symmetry and induction in model checking," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1996.

[9] A. Miller, A. Donaldson, and M. Calder, "Symmetry in temporal logic model checking," ACM Comput. Surv., vol. 38, no. 3, p. 8, 2006.

[10] A. Donaldson and A. Miller, "A computational group theoretic symmetry reduction package for the spin model checker," Lecture Notes in Computer Science, vol. 4019, pp. 374–380, September 2006.

[11] E. A. Emerson and R. J. Trefler, "From asymmetry to full symmetry: New techniques for symmetry reduction in model checking," in In Conference on Correct Hardware Design and Verification Methods. Springer, 1999, pp. 142–156.

[12] A. P. Sistla and P. Godefroid, "Symmetry and reduced symmetry in model checking," ACM Trans. Program. Lang. Syst., vol. 26, no. 4, pp. 702–734, 2004.

[13] S. Barner and O. Grumberg, "Combining symmetry reduction and under-approximation for symbolic model checking," Form. Methods Syst. Des., vol. 27, no. 1-2, pp. 29–66, 2005.

[14] R. Bakhshi and D. Gurov, "Verification of peer-to-peer algorithms: A case study," Electron. Notes Theor. Comput. Sci., vol. 181, pp. 35–47, 2007.

[15] T. Lu, S. Merz, and C. Weidenbach, "Model Checking the Pastry Routing Protocol," in 10th International Workshop Automated Verification of Critical Systems, J. Bendisposto, M. Leuschel, and M. Roggenbach, Eds. Düsseldorf, Allemagne: Universität Düsseldorf, Sep. 2010, pp. 19–21, short communication.

[16] R. Heckel, "Stochastic analysis of graph transformation systems: A case study in p2p networks," in Theoretical Aspects of Computing – ICTAC 2005, ser. Lecture Notes in Computer Science, D. Van Hung and M. Wirsing, Eds. Springer Berlin / Heidelberg, 2005, vol. 3722, pp. 53–69.

[17] P. D. E M Clarke, Grumberg O., Model Checking. The Mit Press, 1999.

[18] R. Cavada, A. Cimatti, C. A. Jochim, G. Keighren, E. Olivetti, M. Pistore, M. Roveri, and A. Tchaltsev, NuSMV 2.4 User Manual, 2005.

[19] L. Zhao, J.-G. Luo, M. Zhang, W.-J. Fu, J. Luo, Y.-F. Zhang, and S.-Q. Yang, "Gridmedia: A practical peer-to-peer based live video streaming system," in Multimedia Signal Processing, 2005 IEEE 7th Workshop, Oct 2005, pp. 1–4.