Model checking the convergence property of BGP networks

Ping Yin[a], Yinxue Ma[b], Zhe Chen[b]
[a] School of Science, Jiangnan University, Wuxi 214122, China
Email: wanderapple@gmail.com
[b] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
Email: nuaacpp@126.com, zhechen@nuaa.edu.cn

***Abstract*— The Border Gateway Protocol (BGP) is an important inter-domain routing protocol, which is widely used in Internet. It allows independent policies to be designed for each Autonomous System (AS). However, the flexibility in designing independent policies causes the convergence problem, i.e., a BGP network may constantly send routing information between ASes and cannot reach a stable state. In this paper, we propose an approach for model checking the convergence property of BGP networks. We firstly establish a formal model of BGP networks and define its convergence property. Then we use the Promela language to describe this model and analyze its convergence. The model is generic enough, thus different instances of BGP networks can be simulated and verified by only modifying parameters and policies. Finally, as examples, we simulate and verify some typical instances of BGP networks by using the SPIN model checker.**

***Index Terms*— border gateway protocol, BGP, model checking, verification, formal methods, SPIN**

I. Introduction

THE Border gateway protocol (BGP) [1] is an important inter-domain routing protocol. It involves sending routing information between different Autonomous Systems (ASes). BGP allows policies which are independently designed for each AS. Thus, there would exist conflicts between these policies. For example, whether the protocol is convergent is an important security problem. Due to the complexity of Internet, the convergence verification of BGP networks should be done with the aid of computers.

Model checking is an important automatic verification technique which has been successfully applied to hardware and software verification. Model checking is performed automatically to verify its properties. If the properties are not satisfied, it provides a counter example. SPIN [2] [3] is an important model checker which has been widely applied in industrial [4] [5]. It is easy to use the Promela language and linear temporal logic (LTL) formulas to describe concurrent systems and properties respectively. Therefore, SPIN is a tool suitable for verifying communication protocols [6]. Other verification techniques are also studied in the literature [7] [8] [9] [10].

Model checking has already been used in formal analysis of network protocols. In [11], the authors used the theorem-proving tool HOL and the model checker SPIN to analyze the key properties of Routing Information Protocol (RIP) and Ad-hoc On-demand Distance Vector (AODV) routing protocol. In [12], the authors studied security vulnerabilities of OSPF with the model checker CBMC. There are also some existing studies on checking the convergence of protocols in the literature. In [13], it is proved that independent design of BGP policies in different ASes causes route oscillations. In [14], the authors analyzed the convergence of BGP with static analysis and gave its computational complexity. In [15] [16], the authors defined a simplified model SPVP to analyze the convergence. We refer to [17] [18] [19] [20] for the theoretical and experimental analysis of BGP convergence in detail.

However, these related works have two drawbacks. First, the proposed theories are too abstract to be easily understood and used in practice, although they contribute to the theoretical aspect. For example, the works [14]–[16] focused on the theoretical proofs of the complexity results of verification problems, instead of experimental results. Some other works [21] [22] focused on the Maude tool, whose language is hard to be understood by average engineers. Second, the proposed models are designed for specific BGP instances with some specified policies. Thus, it is not easy to adapt the existing work, when people have another BGP instance to be verified. We believe that our work can improve these two weaknesses.

In this paper, we propose a verification approach for analyzing the convergence property of BGP networks using model checking. Firstly, we make an abstraction of BGP networks, build a general model and define the convergence property of this model. Then, we use the Promela language to describe this model, whose convergence property is specified by LTL formulas. As examples, we apply this model to three typical BGP instances with different presentations of the convergence property. Finally, SPIN is successfully used to simulate

Manuscript received August 10, 2011; revised January 2, 2012; accepted April 16, 2012.

This work is supported by the National Nature Science Foundation of China (Grant Nos. 11226307 and 61100034), the Scientific Research Foundation for the Returned Overseas Chinese Scholars of State Education Ministry, and the Fundamental Research Funds for the Central Universities (Grant No. JUSRP 11213).

doi:10.4304/jsw.9.6.1619-1625

the model and verify the convergence property.

Comparing with related works, our major contribution is constructing an intuitive and general enough BGP model described by the Promela language. First, our model is easy to be understood by average engineers, thus can be used to verify their BGP instances at hand. Second, our model is also general and flexible, thus the model can be easily adapted to simulate and verify different BGP instances by modifying node parameters and policies. Therefore, our model can be used as a powerful tool for designing correct BGP networks.

The paper is organized as follows. Section II introduces BGP networks, the Promela modeling language in SPIN, and LTL formulas which are used to describe properties. Section III abstracts BGP and builds a formal model which concerns policies and the convergence property. Section IV gives a Promela model based on the abstract model proposed in Section III, as well as an introduction to its convergence property in SPIN. Section V shows three experiments on verifying BGP networks which are proven to be convergent, divergent and partial convergent respectively. Besides, we briefly explain how to verify the networks which are topologically instable. Section VI concludes this paper and proposes future works.

II. Preliminary

A. BGP

BGP is an inter-domain protocol which enables passing routing information between ASes. A BGP network sends the whole routing table when the network initializes. Afterwards, it only sends route changes by route announcements.

A route announcement consists of network layer reachability information (NLRI) and members describing how to reach NLRI such as AS_path, Next_hop, Local_pref. Here, AS_path means a list of ASes, i.e., a path to the destination. If there are two equal AS numbers in the path, then there is a loop. Next_hop means the next hop in the path AS_path, denoted by an AS number. Local_pref means the local priority used in a local AS. The higher the value for a path is, the more preferable the path is.

While a BGP router receives a route announcement from a peer neighbor, it firstly uses its import policy to modify routing information (e.g. set a value to Local_pref) and decides whether the announcement should be added to the Route Information Base (RIB) Adj-RIBs-In. Then the router selects the best route by using the local route strategy and adds the route to Loc-RIB. If the selected best route is different with the previous one, the new route will be added to Adj-RIBs-Out. Then the new route is modified by the export policy, and announced to all peers of the router.

B. Model checker SPIN

In order to verify BGP networks by using SPIN, we firstly use the Promela language to build a model describing their behaviors. Then LTL is applied to describe the desired properties. Finally SPIN would automatically verify these properties. If the properties are not satisfied, SPIN provides counter examples. In this section, we will give a brief introduction to Promela and LTL. More details could be found in the literature [4] [3].

1) Promela: A Promela program usually includes data objects, processes and message channels. Declarations of data objects are rather similar to the C language. Basic data types include bool (or bit), byte (integers from 0 to 255) etc. Other data structures include arrays and records (using the keyword typedef) etc. There are two kinds of scopes for data objects: global and local. There are also macro definitions, inline function definitions in Promela. The body of an inline function is used to replace the call expressions.

Processes may share information via channels and global variables. A process type $pname$ could be declared as follows:

```
proctype pname(list of parameters)
 { body }
```

The initial process $init$ is a process without any parameters. It instantiates other process types by running the process. For example,

```
init
{
run pname(...);
  /*pname is ran as a process*/
}
```

Channel buffers are declared with the keyword chan. The size of a channel varies from 1 to 255, i.e., a channel can store 1 to 255 messages. A channel is used to simulate asynchronous message passing. For example, a channel which is capable of storing up to five messages consisting of two date types (byte and bool) can be defined as follows:

```
chan ch = [5] of { byte, bool }
```

The basic forms of sending and receiving messages are defined as follows:

```
ch ! 6, true; /* send 6 and true. */
ch ? x, bx;   /* receive and assign them
              to variables x and bx. */
```

2) LTL: LTL is obtained by adding temporal operators to propositional logic. An LTL formula is assigned a Boolean value based on state sequences (i.e. paths). LTL is suitable for describing properties of concurrent systems, such as safety properties and liveness properties. We will give a part of the grammars and semantics of LTL that will be used in the sequel.

An LTL formula is

- an atomic proposition denoted by letters, e.g., p, q, and
- if p and q are LTL formulas, then $p \wedge q$, $p \vee q$, $\neg p$, $p \rightarrow q$, pUq (until), $\Box p$ (always), $\Diamond q$ (eventually) are also LTL formulas.

The semantics of these temporal operators are as follows. $\Box p$ denotes that the proposition p is always satisfied through all states in an execution sequence. $\Diamond p$ means that p is eventually satisfied in a state in an execution sequence. For example, the formula $\Box\Diamond p$ represents that, for all the states in an execution sequence, there will be eventually a succeeding state satisfying p. The formula $\Diamond\Box p$ represents that, there is eventually a state whose succeeding states always satisfy p.

III. An abstract model of BGP

In this section, we will build an abstract BGP model and describe its convergence property.

A. Simplification and assumption of BGP networks

In order to focus on the convergence property, we would like to simplify BGP networks as follows:

- Network topology is represented by an undirected graph. A node in the graph, representing an AS, is denoted by a natural number.
- There is only one destination node in a network topology, denoted by 0. This abstraction is sound, thanks to the symmetry of network topologies.
- The communication between ASes is bidirectional.
- A route message consists of AS_path, Next_hop, and Local_pref. The route announcement transferred between ASes only contains AS_path.
- Each AS maintains a routing table RIB storing available routes.
- There is one and only one best route in RIB, denoted by current_best_route.

B. Definition of BGP networks

An undirected graph $G = (V, E)$ represents a network topology, where a vertex $v \in V (0 \leq v < n)$ is an integer denoting a node in the network, $e = (v, w) = (w, v) \in E$ is an edge between neighbor vertices v and w, and $d = 0$ is the unique destination node which only sends route information to neighbor nodes at the initial stage.

A BGP network is represented by a quadruple $S = (G, P, \Lambda, M)$, where G is an undirected graph describing a network topology, $P = \cup\{P_v | v \in V\}$ is a set of permitted paths, $\Lambda = \{\lambda_v | v \in V \setminus \{0\}\}$ is a set of local preference assignment functions where λ_v is a function for assigning priorities, and M is a channel buffer which is used to store route announcements. A router can send a route message to the buffer, and receive a message from the buffer.

C. Operations on nodes

Node v receives a route announcement from a message channel, modifies the message by using the import policy and updates RIB_v. Then the best route is selected from RIB_v by calling the route selection process. If the new best route is not the same as the old one, node v will send the route announcement to all its neighbor nodes. The details are explained as follows.

1) **Import policies**: Node v receives a route message r with $r.AS\_path = w, \cdots, d$ from its neighbor w. Then it updates $r.AS\_path = v, w, \cdots, d$, and checks whether path $r.AS\_path$ belongs to P_v. If true, then let $r.Next\_hop = w$, $r.Local\_pref = \lambda_v(r)$.
2) **Updating** RIB_v: If the updated path $r.AS\_path$ does not belong to P_v (e.g. there is a cycle in $r.AS\_path$), v will discard the path from w and check RIB_v. If there is a path r' which is an old path from w to d, then r' will be discarded as well, that is, $RIB_v = RIB_v \setminus r'$. Otherwise, v accepts r and checks RIB_v. Similarly, if there is a path $r' = w, \cdots, d$, which is different from r, then r' will be discarded from RIB_v, and r will be added to RIB_v.
3) **Local route selection policies**: We select the best route from the updated route table RIB_v of node v. Two different routes are compared in each round until the comparison of all routes in RIB_v finishes. Therefore the new best route is selected. There are three rules for route comparison as follows. Assume two arbitrary route information $r1$ and $r2$ in RIB_v.
   - If $r1.Local\_pref \neq r2.Local\_pref$, choose the one with the higher value of $Local\_pref$.
   - Otherwise, if the lengths of path $r1$ and $r2$ are different, choose the one with the shorter path length.
   - Otherwise, choose the one with the lower value of $Next\_hop$.
4) **Updating and sending** $current\_best\_route$: If the selected best route is different from the old one, the new one is assigned to $current\_best\_route$. Meanwhile, the node v sends a route announcement including $current\_best\_route$ to all its neighbors. If the selected best route is the same as the old one, the node v does no operation, waiting for the messages from its neighbors.

D. Formalizing Convergence Property

If all the nodes keep unchanged in an execution, we assume that the system model reaches a stable state. Formally, if a stable state satisfies the following two conditions, we assume the system model is convergent in an execution:

- $M = \emptyset$,
- For any two nodes v and w, if the current best route of node v is $v, w, v_{i1}, \cdots, v_{ik}, d$, the current best route of w is definitely $w, v_{i1}, \cdots, v_{ik}, d$.

The convergence of a system model is determined by the stability in all runs. To describe the system convergence, we consider three cases as follows:

- **Convergent**: if the system model is stable in all runs.
- **Divergent**: if the system model is unstable in all runs.
- **Partially convergent**: if the system model is stable in some runs and unstable in others.

IV. Modeling BGP in Promela

In this section, we propose a BGP model in Promela based on the abstract model in Section III. In this Promela model, every node is a process consisting of a number of inline functions. The parameter N denotes the number of nodes. To simulate different BGP networks, we only need to change the value of N and the statements of some functions related to local preference assignment.

A. Data structures

We firstly give the data structures described using Promela as follows,

```
#define N 4
Typedef EDGE {
  bit b[N];
}
EDGE a[N];

typedef  ROUTE {
  byte path[N];
  byte count;
  byte Local_pref;
}
chan ch = [N*(N-1)-2] of
{ byte, byte, ROUTE.path[N], ROUTE.count };
/* The first byte is the identity of sender
the second byte is the identity of receiver.*/
```

In the model, N denotes the number of nodes. For example, if there is a BGP model of four nodes, we write `#define N 4`. The two-dimensional array a[0,...,N-1].b[0,...,N-1] represents the link state between two nodes. If nodes i and j are linked, then we set a[i].b[j] = a[j].b[i] = 1. Otherwise, a[i].b[j] = a[j].b[i] = 0. ROUTE is a structure containing AS_path, Next_hop and Local_pref, where `count` is the number of nodes in the AS_path, and path[0,...,N-1] is an array storing node numbers. The combination of count and path represents AS_path and Next_hop. For example, the expressions path[0] = 0, path[1] = 2, path[2] = 3, path[3] = 2, count = 3 denote a path 320 from node 3 to destination node 0, Next_hop in this path is path[count-2] = 2. Finally, the variable *ch* is a channel with the buffer of the size N*(N-1)-2. The route messages sent between ASes are stored in *ch*, and the intended receivers take their messages from *ch*.

B. Node processes and the initial process

Based on the data structures, we give the pseudo code of node processes and the initial process in Promela. Each node is an instantiated process, as shown in following program.

```
proctype NODE(byte v)
{
  ROUTE cbr; /*current_best_route*/
  ROUTE nbr; /*new_best_route*/
  if
  :: v == 0 -> /* The only destination node.*/
     atomic{
     cbr.count = 1;
     send(v, cbr);}
  :: else ->
     /* If the node is not destination one. */
     do
     :: msgReceive ->
      receive(msg);
      import(v, msg);
      routeSelectRIB(v, nbr);
      compareRoute(nbr, cbr);
      if
      :: routeNotSame ->
         cbr = nbr;
         send(v, cbr)
      :: else -> skip;
      fi;
     :: else -> skip;
     od;
  fi;
}
```

We fix the destination node to be node 0. It only sends messages to all its neighbors at the beginning. Because all default values of variables in Promela programs are equal to 0, we firstly assign the count of current_best_route to 1. Thus we get the best route of node 0, i.e., AS_path is 0. Then this route information will be sent to the neighbors of node 0 by calling send(v, cbr).

When a node except the destination node receives a message from its neighbors by calling receive(msg), it firstly determines whether the path included in this message msg is permitted to receive, updates RIB, then selects the new best route from RIB and assign it to nbr by calling routeSelectRIB(v, nbr), finally compares the new best route and the current best route by calling compareRoute(nbr, cbr). If they are not the same, the current_best_route is updated by executing cbr = nbr, and the new best route is sent to all neighbors of node v by calling send(v, cbr). Otherwise, nothing is done.

In the initial process, the network topology is generated. Then node processes are instantiated. An example of initial processes which can generate a network topology as a complete graph is shown as follows.

```
init {
  byte i = 0;
  byte j = 1;
  do
  /* Generate a complete graph with N nodes. */
  :: i < N ->
     do
     :: j < N ->
        a[i].b[j] = 1;
        a[j].b[i] = 1;
        j++;
     :: else -> break;
     od;
     i++;
     j = i+1;
  :: else -> break;
  od;
  atomic {
    /*Instantiate process of nodes*/
    i = N;
    do
    :: i > 0 ->
       run NODE(i-1);
       i--;
    :: i == 0 ->
       break;
    od;
  }
```

```
29 }
```

C. Formalizing Convergence in LTL

Now we can describe the convergence property in LTL. Based on the defined and constructed Promela model, the convergence is determined by only checking the state of channel buffers. Two LTL formulas P1 and P2 will be used as follows.

- **P1**: <>[] len(channel) == 0
- **P2**: []<> len(channel) != 0

Note that, in SPIN/Promela, temporal operators $\Box$ and $\Diamond$ are denoted by [] and <> respectively. The function len(chan channel_name) is used for calculating the number of messages in a channel buffer.

P1 means that there eventually exists a state in a run such that the channel buffers of all its succeeding states are empty. P2 means that, for all states in a run, there always exists a succeeding states with nonempty channel buffer in the run.

We perform the convergence analysis of a BGP network by verifying whether counter examples exist for P1 and P2 and combining the verification results in the way shown in Table I. Note that it is impossible that P1 and P2 have no counter examples simultaneously.

TABLE I.
CONVERGENCE CONCLUSION

P1	P2	convergence
no counter example	counter example	convergent
counter example	no counter example	divergent
counter example	counter example	partially convergent
no counter example	no counter example	-

V. VERIFYING THE CONVERGENCE PROPERTY OF BGP NETWORKS USING SPIN

In this section we give examples of BGP networks which are convergent, divergent or partially convergent [15]. They are simulated and verified with SPIN/Promela by changing node parameters and priority functions. We also specify unstable networks.

A. Examples of BGP

Fig 1 and Fig 2 are network topologies which we use in this section. Assume that these topologies are unchanged in a run and node 0 is the only destination node. We give the local priority functions of the three examples as follows. The permitted paths P_v is a set of paths which are loop free from node v to node 0.

- **e1**: $\lambda_v(p) = 0$, $p \in P_v$.
- **e2**: If $p \in \{120, 230, 310\}$, $\lambda_v(p) = 1$. Otherwise, $\lambda_v(p) = 0$.
- **e3**: If $p \in \{120, 210\}$, $\lambda_v(p) = 1$. Otherwise, $\lambda_v(p) = 0$.

We expect that e1 is convergent, e2 is divergent, and e3 is partially convergent. The experimental results in the next subsection are consistent with the expected results.

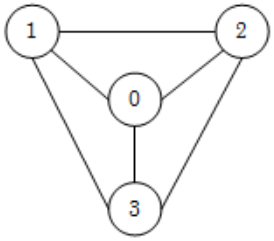

Figure 1. Network topology for examples e1 and e2.

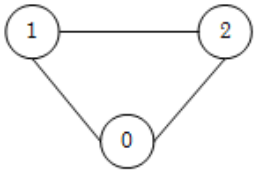

Figure 2. Network topology for example e3.

B. Experimental results

In this section we give the experimental results, memory and time consumptions in Table II. The results are consistent with the expected ones, i.e., e1 is convergent, e2 is divergent and e3 is partially convergent.

The experiments were conducted on a dual-cored laptop with 2GB memory and 2.00GHz CPU. We verified these examples with or without memory compression technologies. Memory compression includes lossless compression (e.g., collapse compression, minimized automaton representation) and lossy compression (e.g., bitstate hashing method, hash compact method).

We found that the verifications with counter examples were finished within one second without using any memory compression technology. If lossless compression was applied, the increment of memory and time consumptions was not obvious. If lossy compression was applied, the memory consumption significantly decreased, whereas the time consumption greatly increased. Moreover, lossy compression cannot be used in e3, because it affects the correctness of verification since counter examples could not be found due to the decreased coverage of the state space. In the case of verifications without counter examples, we find that it is difficult to verify the model without using any memory compression technology, even for a network consisting of 4 nodes, as shown in Table II. Thus, we must complete the verifications using memory compression technologies.

examples	properties	counter examples	no memory compression			lossless compression			lossy compression		
			states	memory (MB)	time (S)	states	memory (MB)	time (S)	states	memory (MB)	time (S)
e1	P1	no	-	-	-	85794462	259.168	5290	25344184	17.515	618
	P2	yes	4335	65.808	0.15	4335	64.636	0.14	6108344	18.101	54.5
e2	P1	no	8759	66.980	0.39	8759	64.832	0.36	24613703	17.808	648
	P2	no	-	-	-	333921150	1678.894	19800	42583137	18.004	350
e3	P1	yes	1392	64.734	0.05	1392	64.539	0.05	no counter example		
	P2	yes	281	64.539	0.01	281	64.539	0.01	no counter example		

TABLE II.
RESULTS OF EXPERIMENTS

The state space is huge even for a 4 nodes network. For example, there are 333921150 states in the verification of property P2 in e2, using lossless compression. And the memory and time consumptions are 1678.894MB and 19800s respectively. It is caused by the state explosion problem which is a bottleneck of model checking.

C. Verification on unstable networks

The above network examples are stable in their topological structures. It means that the failures of nodes and links do not appear in runs. However, the instability of nodes and links can happen in reality. We regard the failures of nodes as the failures of links, since when the node v fails, all links between v and its neighbor nodes fail as well.

When a link failure happens, the nodes connecting to this link would send link failure messages to their neighbors. The system begins a new converging procedure from unstable states to stable states.

Link failures mean that the old topology has changed and a new topology arises. Therefore, we consider link failures as dynamics of topologies. Given N nodes, the dynamics of topologies can be simulated by the automatic generation of a random topology at the beginning.

VI. CONCLUSIONS

In this paper, we proposed a formal model of abstract BGP and its corresponding model in Promela. All BGP networks can be simulated by changing the parameters of nodes and priority functions. To analyze the convergence of BGP networks, we consider three types, i.e., convergent, divergent and partially convergent. We verified the examples of BGP networks of these three types using SPIN, and gave the convergence conclusion.

In the future, we would like to extend this work in the following aspects. First, we will try to improve the efficiency of our model. As we know, the state space explosion problem is a critical bottleneck of applying model checking to large systems [23]. Thus, we may try to use reduction methods and compression methods in our experiments to increase its ability of checking larger networks. Second, we will try to increase the size of our model. As we know, although SPIN is a powerful model checker, it has the limitation on the size of channel buffers (up to 255). As a result, we can only verify the networks with the number of nodes up to 16. We would like to solve this restriction by using more advanced theories about data structures in modeling and their algorithms, such as automata and ω-automata [24]. Third, we may study the equivalent reduction of a large network to a small one, conversion of an unstable network to a special stable network. Besides, we can combine theorem proving with model checking to analyze BGP networks. Fourth, we may investigate other possible methods for verifying the correctness of protocols, e.g., the runtime monitoring approach [25], which is based on the formalism of control systems on automata [26] [27].

ACKNOWLEDGMENT

This work is supported by the National Nature Science Foundation of China (Grant Nos. 11226307 and 61100034), the Scientific Research Foundation for the Returned Overseas Chinese Scholars of State Education Ministry, and the Fundamental Research Funds for the Central Universities (Grant No. JUSRP 11213). The authors gratefully acknowledge the helpful comments and suggestions of the anonymous reviewers, which have greatly improved the presentation of this paper.

REFERENCES

[1] Y. Rekhter, T. Li, and S. Hares, *A border gateway protocol 4 (BGP-4)*, http://tools.ietf.org/html/rfc4271, 2006.
[2] B.-A. Mordechai, *Principles of the SPIN model checker*. Springer, 2008.
[3] G. J. Holzmann, *The SPIN model checker: primer and reference manual*. Addison-Wesley, 2003.
[4] Z. Chen, Y. Gu, Z. Huang, J. Zheng, C. Liu, and Z. Liu, "Model checking aircraft controller software: A case study," *Software-Practice & Experience*, 2014. Available at http://dx.doi.org/10.1002/spe.2242.
[5] Z. Chen and G. Motet, "Methodology and experience for designing safety-related systems in iec 61508," in *Proceedings of the 4th International Conference on Dependability (DEPEND 2011)*. IARIA, 2011, pp. 57–64.
[6] Z. Chen, D. Zhang, R. Zhu, and et al., "A review of automated formal verification of ad hoc routing protocols for wireless sensor networks," *Sensor Letters*, vol. 11, no. 5, pp. 752–764, 2013.
[7] H. Shi, W. Ma, M. Yang, and X. Zhang, "A case study of model checking retail banking system with spin," *Journal of Computers*, vol. 7, no. 10, pp. 2503–2510, 2012.
[8] B. Meng, W. Huang, and Z. Li, "Automated proof of resistance of denial of service attacks using event with theorem prover," *Journal of Computers*, vol. 8, no. 7, pp. 1728–1741, 2013.

[9] Z. Wei, C. Xia, Y. Luo, X. Liu, and W. Wu, "An approach for description of computer network defense scheme and its simulation verification," *Journal of Computers*, vol. 9, no. 2, pp. 388–395, 2014.

[10] G. Zhao and X. Li, "Identity authentication scheme expansion based on speaker verification," *Journal of Computers*, vol. 8, no. 8, pp. 2027–2033, 2013.

[11] K. Bhargavan, D. Obradovic, and C. A. Gunter, "Formal verification of standards for distance vector routing protocols," *Journal of the ACM*, vol. 49, no. 4, pp. 538–576, 2002.

[12] A. Sosnovich, O. Grumberg, and G. Nakibly, "Finding security vulnerabilities in a network protocol using parameterized systems," in *Proceedings of the 25th International conference on Computer Aided Verification*, 2013, pp. 724–739.

[13] R. G. K. Varadhan and D. Estrin, "Persistent route oscillations in inter-domain routing," *Computer Networks*, vol. 32, no. 2, pp. 1–16, 2000.

[14] T. G. Griffin and G. Wilfong, "An analysis of bgp convergence properties," in *Proceedings of the ACM International conference on the applications, technologies, architectures, an-d protocols for computer communication*, 1999, pp. 277–288.

[15] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "Policy disputes in path-vector protocols," in *Proceedings of the seventh international conference on network protocols*, 1999, pp. 21–30.

[16] ——, "The stable paths problem and interdomain routing," in *IEEE/ACM transactions on networking*, vol. 10, no. 2, 2002, pp. 232–243.

[17] J. L. Sobrinho, "Network routing with path vector protocols: theory and applications," in *Proceedings of the ACM international conference on the applications, technologies, architectures, and protocols for computer communication*, 2003, pp. 49–60.

[18] N. Feamster and H. Balakrishnan, "Detecting bgp configuration faults with static analysis," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, 2005, pp. 43–56.

[19] L. J. Wang, J. P. Wu, and K. Xu, "Analysis of bgp convergence using shlpn model," in *Proceedings of the third advanced international conference on telecommunications*, 2007.

[20] F. Le, S. Lee, and T. Wong, "Detecting network-wide and router-specific misconfigurations through data mining," in *IEEE/ACM Transactions on Networking*, vol. 17, no. 1, 2009, pp. 66–79.

[21] A. Wang, C. L. Talcott, and L. M. Jia, "Analyzing bgp instances in maude," *Dingel R B J. Formal Techniques for Distributed Systems*, pp. 334–348, 2011.

[22] A. Wang, C. Talcott, and A. J. Gurney, "Reduction based formal analysis of bgp instances," *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 7214, pp. 283–298, 2012.

[23] Z. Chen and G. Motet, "Nevertrace claims for model checking," in *Proceedings of the 17th International SPIN Workshop on Model Checking of Software (SPIN 2010)*, ser. Lecture Notes in Computer Science, vol. 6349. Springer, 2010, pp. 162–179.

[24] Z. Chen, "On the generative power of ω-grammars and ω-automata," *Fundamenta Informaticae,*, vol. 111, no. 2, pp. 119–145, 2011.

[25] Z. Chen and G. Motet, "System safety requirements as control structures," in *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2009)*. IEEE Computer Society, 2009, pp. 324–331.

[26] Z. Chen, "Control systems on automata and grammars," *The Computer Journal*, 2014. Available at http://dx.doi.org/10.1093/comjnl/bxt125.

[27] Z. Chen and G. Motet, "Towards better support for the evolution of safety requirements via the model monitoring approach," in *Proceedings of the 32nd International Conference on Software Engineering (ICSE 2010)*. ACM, 2010, pp. 219–222.

Ping Yin is an assistant professor in school of science at Jiangnan University. Her main research interests include numerical computing, mathematical software, and system and software verification. She got the Ph.D. degree in applied mathematics from Aix-Marseille University (AMU), France in 2011.

Yinxue Ma is a master student of computer science at Nanjing University of Aeronautics and Astronautics. Her research interests include formal methods, software verification and their applications to the design of network protocols.

Zhe Chen is an associate professor of computer science at Nanjing University of Aeronautics and Astronautics. He received his Ph.D. degree in Computer Science from Institut National des Sciences Appliquees (INSA) de Toulouse, Universite de Toulouse, France in 2010. His research interests include formal methods, software verification, programming and modeling languages, and their applications to the design of network protocols and industrial systems. He has published over 20 peer-reviewed research papers in major journals and conferences such as SPE, COMPJ, FI, ICSE, SPIN and COMPSAC. He is the author of three books on programming. He is a member of CCF, ACM and IEEE.