社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－CSEC－25　(9)
2004／5／21

タグスイッチネットワークのモデル検査による漏洩検査

櫻　田　英　樹[†]

タグに従ってパケットが転送されるネットワークでは, ノードの設定ミスにより, パケットが予期しないセグメントに漏れ出したり, 目的のセグメントに到達しないなどのトラブルが生じることがある. パケットの漏れ出しは特に, セキュリティを脅かす問題である. 本論文では, タグに従ってパケットが転送されるネットワークをモデル化し, モデル検査によってパケットの到達性を検査する方法について述べる. モデルは記号モデル検査のツールである NuSMV 上で実装される. この方法はノードの設定が実際に用いられる前にその正しさを確認するためや, トラブルが起きたときに原因を究明するために有用である.

Model Checking Configurations for Tag-Switched Computer Networks

Hideki Sakurada[†]

In tag-switched computer networks, errors in the configurations of the nodes can cause the frames to leak into unexpected segments or not to reach the destination segments. Frame leakage, in particular, can be a security problem. This paper describes a method to model tag-switched networks and check the reachability of frames by model checking. The model is implemented on NuSMV, a tool for symbolic model checking. The method is useful for checking configurations before they are activated and for analyzing the cause of trouble.

1. Introduction

Managing a computer network is more or less a difficult job because the network is a distributed system that consists of nodes (routers, switches, or terminals) that work independently. Changing the functionalities of a part of a network requires one to consistently change the configurations of more than one nodes at the same time. If one forgets to change one of them, the part does not work as intended and even other parts of the network may work incorrectly. Moreover, the symptoms of the trouble often appear after a few hours from the change. Because it is sometimes hard to determine the effect of configuration changes or the cause of the trouble, applying formal methods to network management is appealing.

This paper focuses on tag-switched networks. A tag-switched network is a network where data are conveyed in frames with tags. The nodes that construct the network forward frames from a port to selected ports according to the tags attached to the frames. Tag-switched networks are commonly used to make more than one LANs coexist in a single physical network. The LANs are called virtual LANs (VLANs). Figure 1 shows a simple tag-switched network that contains two VLANs and

[†] 日本電信電話株式会社 NTT コミュニケーション科学基礎研究所
NTT Communication Science Laboratories, NTT Corporation

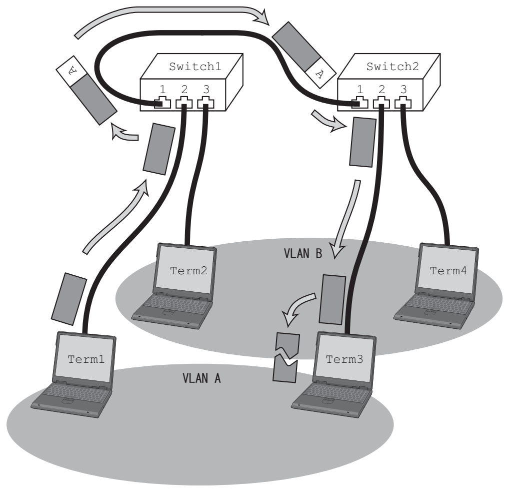

図 1　A simple tag-switched network that contains two VLANs the flow of a frame from `Term1` to `Term3`. The two switches, `Switch1` and `Switch2`, are configured so that they attach tag `A` (tag `B`, resp.) to the frames from port `2` (port `3`, resp.) and forward them to port `1` and that they inversely switch the frames with tag `A` (tag `B`, resp.) from port `1` to port `2` (port `3`, resp.) and remove the tag. Thus, the frame from `Term1` is attached tag `A` by `Switch1`, transmitted to port `1` on `Switch2`, switched to port `2` according to the tag, stripped of the tag by `Switch2`, and finally transmitted to port `1` on `Term3`, which is in the same VLAN as `Term1` is.

Since tag-switching provides a kind of routing mechanism, errors in the configurations of the nodes cause the frames to leak into unexpected segments or not to reach the destination segments. In particular, frame leakage may break the security of the network. This paper describes a method to model tag-switched networks and to assure the reachability of the frames by model checking. The method can be used to check network configurations before they are activated and analyze the cause of trouble. More specifically, the method models a tag-switched network as a Kripke structure and checks the frame reachability specified as CTL formulae. The model allows each frame to have more than one tags to analyze networks based on some standards that are revewed in the next section. The model checking is done with the NuSMV[1)], a matured tool for symbolic model checking. Applying model checking to check configurations for tag-switched networks has not yet been done as far as the author knows. Even if writing a program to assure the reachability directly with graph algorithms is not very difficult, use of a matured tool for model checking enables one to avoid the risk of bugs in the programs. Moreover, since a logic (e.g. CTL[2)] or LTL[3)]) used to specify network properties is quite expressive, one can check a variety of properties other than reachability without any additional modeling.

The rest of this paper is organized as follows. The next section reviews a few standards for tag-switched networks. Section 3 models a simple network without tags. Section 4 extends the model to tag-switched networks and shows an example that models the network shown in Fig. 1. Although our method is explained using only a few simple examples, the method can be easily applied to more complex networks. Section 5 summarizes the results of the paper and discusses some related works.

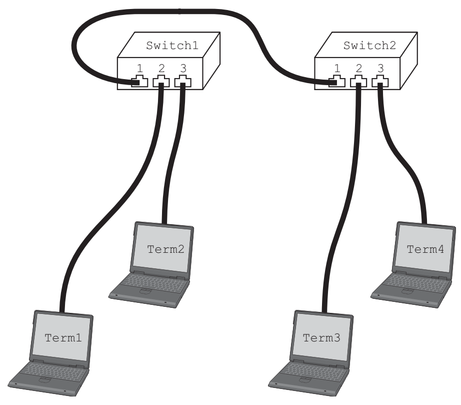

図 2 A simple network without tags

2. Standards for Tag-Switched Networks

IEEE standard 802.1Q[4)] is a standard for tag-switched networks. It extends the header format of the frames of Ethernet or FDDI with a 12-bit tag. This standard is implemented on many so-called "managed layer-2 switches" and is one of the most common ways to construct VLANs in corporate networks. Although the standard allows only one tag on each frame, some vendors independently extend the standard to enable multiple tags on a frame[5)]. These extensions make it possible to construct sub-VLANs in a VLAN without caring about conflicts among the set of the tags.

Multi-protocol Label Switching (MPLS)[6)] is another standard for tag-switched networks. MPLS allows each frame to have a stack of tags (labels) after the header of the frame. When a node on a MPLS network receives a frame, it decides the next-hop node according to the tag on top of the stack. The node optionally pushes, pops, and alters a tag on top of the stack. MPLS is widely used by carriers in IP-VPN services to avoid conflicts in the private IP address spaces of their customers and to identify traffic that needs different qualities of services.

3. A Simple Network Model

In this section, a simple network without tags is modeled. The network is shown in Fig. 2. It has the same physical structure as the network in Fig. 1. The nodes in the network receive frames from any port and forward them to all the ports other than the incoming port.

Since the flows of the frames are of interest, the

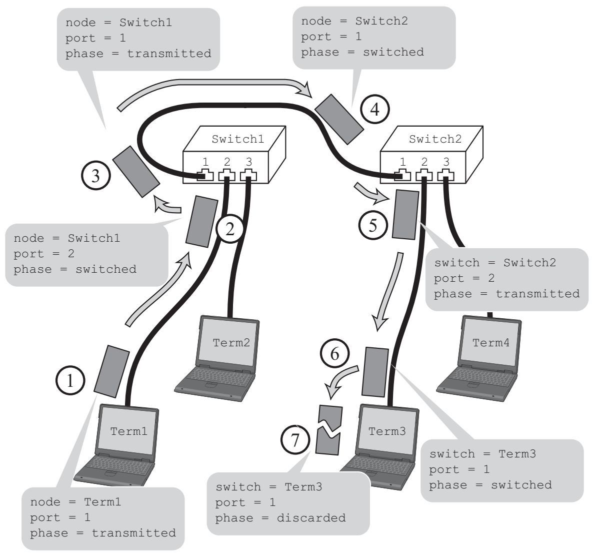

図 3 The trace of the state transition for the flow of a frame

model is given as a Kripke structure where each state represents the physical location and the internal state of a frame. The Kripke structure has three variables: `node`, `port`, and `phase`. The first two represent the physical location of a frame. The last one represents the internal state of a frame and the value is `switched`, `transmitted`, or `discarded`. They respectively indicate that the frame is switched from a port to another port inside a node, transmitted from a port of a node to a port of another node through a cable, or discarded on a port on a node. For the network considered in this section, the value of `node` is `Switch1`, `Switch2`, `Term1`, `Term2`, `Term3`, or `Term4` and the value of `port` is an integer between `1` and `4`. A trace that represents a flow of a frame from `Term1` to `Term3` is shown in Fig. 3. In the first state of the trace, the frame is on port `1` on `Term1` and is to be transmitted to the peer port. In the second state, the frame is on port `2` on `Switch1`, which is the peer port in the previous state, and is to be switched inside the switch. In the third state, the frame is switched to port `1` on the same switch and is to be transmitted to the peer port. Note that a trace where the frame is switched to port `3` or `4` is also possible here. After the interleaving transmissions and switchings, the frame reaches port `1` on `Term3`. And then the frame is received by the terminal and discarded on the port of the switch in the final state.

The script that implements the model on NuSMV is shown in Fig. 4. The `VAR` section de-

```
MODULE main
VAR
  node: {Switch1, Switch2,
         Term1, Term2, Term3, Term4};
  port: 1..3;
  phase: {switched, transmitted, discarded};
TRANS
  case
    phase = switched:
      case
        node in {Switch1, Switch2}:
          case
            port = 1:
              next(port) in {2,3}
              & TRANSMIT;
            port = 2:
              next(port) in {1,3}
              & TRANSMIT;
            port = 3:
              next(port) in {1,2}
              & TRANSMIT;
          esac;
        node in {Term1, Term2, Term3, Term4}:
          DISCARD;
      esac;
    phase = transmitted:
      case
        /* Switch1/1 - Switch2/1 */
        node = Switch1 & port = 1:
          next(node) = Switch2
          & next(port) = 1 & SWITCH;
        node = Switch2 & port = 1:
          next(node) = Switch1
          & next(port) = 1 & SWITCH;
                  ⋮
       /* otherwise */
        TRUE: DISCARD;
      esac;
    phase = discarded:
      DISCARD;
  esac
```

図 4 Implementation of a simple network on NuSMV

clares the variables of the Kripke structure. The `TRANS` section defines the transition relation of the Kripke structure. The expressions `next(node)`, `next(port)`, and `next(phase)` respectively represent the values of the variables `node`, `port`, `phase` in the next state. The `case` expression

`case`
 $cond_1$`:` exp_1

```
  cond₂: exp₂
       ⋮
  condₙ: expₙ
esac
```

returns the value of exp_i if and only if $cond_i$ evaluates to TRUE and all $cond_1,\ldots,cond_{i-1}$ evaluate to FALSE. The transition relation consists of three cases according to the value of phase. The case for switched is generated from the configurations of the nodes. The variables node and port is tested and used to decide the port next(port) in the next state. In Fig. 4, the frames incoming from any port are switched to all ports other than the incoming port by Switch1 and Switch2. The terminals discard the incoming frames. The case for transmitted is generated from the physical construction of the network. The next(port) on next(node) are respectively the peer port on the peer node of port on node. The case for discard is always the same. The next state is same as the current state. The expressions TRANSMIT, SWITCH, and DISCARD are defined by the following macros of the C preprocessor [☆].

```
#define TRANSMIT \
  (next(phase) = transmitted)
#define SWITCH \
  (next(phase) = switched)
#define DISCARD \
  (next(phase) = discarded & \
   next(port) = port & \
   next(node) = node)
```

The requirements for the network can be specified as CTL formulae. For example, the reachability of the frame originating from port 1 on Term1 to port 1 on Term3 is specified by the following formula:

```
SPEC
  (node = Term1
   & port = 1
   & phase = transmitted)
  -> EF (node = Term3).
```

The -> and EF are respectively implication and the "exists eventually" modal operator.

4. Extension with a Tag Stack

In this section, the network in Fig. 1 is modeled by extending the model in the last section with tags. The Kripke structure to model this network is obtained by extending the structure presented in the last section with a variable ts, a tag stack. The value of ts is a list of tags. The trace of the state transition for the flow of the frame in Fig. 1 is shown in Fig. 5. The assignments of the other variables are the same as in Fig. 3.

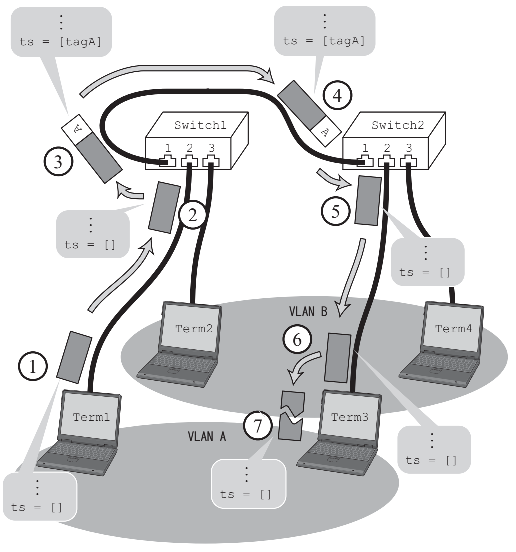

図 5 Trace of the state transition for the flow of the frame in Fig. 1

This model is implemented on NuSMV by the program shown in Fig.6. The variable ts is of type tagstack, whose operations are given as the macros TEST, POP and PUSH. The macro TEST(ts, 1, A) is true if and only if tagA is the 1st tag from the top of the stack ts. The macros POP and PUSH respectively pop and push a tag on top of the stack (i.e. let the pushed or poped stack be the stack in the next state). In the case of phase = switched, the stack of the frame is tested and optionally POP-ed or PUSH-ed when switched. The other part is the same as in Fig. 4.

The detailed implementation of the module tagstack is shown in Fig. 7. Since NuSMV does not have a type for lists, the stack is implemented as an array of a fixed length. The length is three in Fig. 7.

The reachability of the frames is specified in CTL. For example, to make sure that frames originating from VLAN A do not leak to VLAN B, it is sufficient to write the following formula and to check it on NuSMV:

```
SPEC
  port = 1 & node in {Switch1, Switch3}
  -> AG (! (node in {Switch2, Switch4})).
```

The operators AG and ! are respectively the "forall

[☆] NuSMV can be used with the C preprocessor by invoking with the option -cpp

```
MODULE main
VAR
  node: {Switch1, Switch2,
         Term1, Term2, Term3, Term4};
  port: 1..3;
  phase: {switched, transmitted, discarded};
  ts: tagstack;
TRANS
  case
    phase = switched:
      case
        node in Switch1, Switch2:
          case
            port = 1 & TEST(ts, 1, A):
              next(port) = 2 & POP(ts)
              & TRANSMIT;
            port = 1 & TEST(ts, 1, B):
              next(port) = 3 & POP(ts)
              & TRANSMIT;
            port = 2:
              next(port) = 1 & PUSH(ts, A)
              & TRANSMIT;
            port = 3:
              next(port) = 1 & PUSH(ts, B)
              & TRANSMIT;
          esac;
        node in {Term1, Term2, Term3, Term4}:
          DISCARD;
      esac;
    phase = transmitted:
      case
        /* Switch1/1 - Switch2/1 */
        node = Switch1 & port = 1:
          next(node) = Switch2
          & next(port) = 1 & SWITCH;
        node = Switch2 & port = 1:
          next(node) = Switch1
          & next(port) = 1 & SWITCH;
                    ⋮
        /* otherwise */
        TRUE: DISCARD;
      esac;
    phase = discarded:
      DISCARD;
  esac
```

図 6 Implementation of the Extended Network on NuSMV

```
#define MAX 3

#define PUSH(ts, t) \
  ((case \
      ts.size != MAX:  \
        next(ts.overflow) = ts.overflow \
        & next(ts.size) = ts.size + 1; \
      ts.size = MAX: \
        next(ts.overflow) = TRUE & \
        next(ts.size) = ts.size; \
    esac) & \
   next(ts.stack[1]) = t & \
   next(ts.stack[2]) = ts.stack[1] & \
   next(ts.stack[3]) = ts.stack[2])

#define POP(ts) \
  (next(ts.overflow) = ts.overflow & \
   next(ts.size) = ts.size - 1 & \
   next(ts.stack[1]) = ts.stack[2] & \
   next(ts.stack[2]) = ts.stack[3])

#define STAY(ts) \
  (next(ts.overflow) = ts.overflow &  \
   next(ts.size) = ts.size &  \
   next(ts.stack[1]) = ts.stack[1] &  \
   next(ts.stack[2]) = ts.stack[2] &  \
   next(ts.stack[3]) = ts.stack[3])

#define TEST(ts, p, tset) \
  ((p <= ts.size) & (ts.stack[p] = tset))

MODULE tagstack
VAR
  stack: array 1 .. MAX of {tagA, tagB};
  size: 0 .. MAX;
  overflow: boolean;
INIT
  overflow = FALSE
```

図 7 Implementation of the module for a tag stack

globally" modal operator and negation.

The frames on the network handled so far in this section have at most only one tag. Since the model allows any number of tags, although the number is limited in implementation, it is easy to extend the network to include a hierarchy of VLANs. Moreover, the properties that can verify are not limited to reachability because of the expressiveness of CTL and LTL. For example, to make sure that there is no loop of frames in the network, it is sufficient to check that all frames will be eventually discarded, which is specified in the following:

```
SPEC
  EF (phase = discarded).
```

Note that even if the network does not include a physical loop, the frames may loop because of tag switching. For another example, in some larger network, to verify that all frames originating from the node `Node1` to the node `Node2` go through a node `NodeX` that can filter viruses in frames, it is sufficient to write the following specification in LTL:

```
LTLSPEC
  node = Node1
  -> ! (node != NodeX U node = Node2).
```

The operator `U` is the "until" modal operator.

5. Conclusion

This paper has described a method to model tag-extended networks as Kripke structures and check the correctness of the configurations of the nodes that construct the networks by the NuSMV model checker. The method can, for example, check the reachability and the loop freeness of the frames. If the physical construction of a network is given in a machine-readable form, it is easy to automatically generate the NuSMV script for the model from the configurations for the nodes. The author has been developing a tool that collects the configurations from switches via the TFTP protocol or the SNMP protocol and automatically checks the reachability of the frames. Even if writing a program that does the same checks by graph algorithms is not difficult, the method enables one to avoid the risk of bugs in the program. Moreover, by usnig NuSMV or other model checkers that can check specifications written in certain logics, properties of the network other than reachability can be checked with the method.

There are a few works that apply formal methods to network management. Guttman[7] defines a language to model IP networks that consist of the packet filtering gateways and their security policies in terms of reachability. He also gives algorithms to generate the configurations of the packet filters on each interface of the gateways and verify that the policies are realized. He does not use a generic model checking tool for verification in contrast with the present method that uses the NuSMV model checker and allows one to verify properties other than reachability. Guttman *et al.*[8] modeled networks that consist of security gateways that perform IPsec operations on each packet that goes thorough the gateways. The operations include push and pop of an IPsec header on the stack of a frame. They give the constraints on the behaviors of the gateways and prove that the authentication and confidentiality goals are achieved under the constraints. Their model and the present model similarly allow a packet, which is called a frame in this paper, to have a stack of information used by gateways. They focus on the formalization and the investigation of IPsec in contrast with this paper that focuses on checking of running networks.

参 考 文 献

1) Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R. and Tacchella, A.: NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, Lecture Notes in Computer Science, Vol. 2404, Springer, pp. 359–364 (2002).

2) Clarke, E.M., Emerson, E.A. and Sistla, A.P.: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications, *ACM Transaction on Programming Languages and Systems*, Vol. 8, No. 2, pp. 244–263 (1986).

3) Pnueli, A.: A temporal logic of concurrent programs, *Theoretical Computer Science*, Vol. 13, pp. 45–60 (1981).

4) IEEE: IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks (2003). IEEE Std 802.1Q-2003.

5) Riverstone Networks: *RapidOS Management Center User Guide Release 1.1* (2002).

6) Rosen, E., Viswanathan, A. and Callon, R.: Multiprotocol Label Switching Architecture (2001). RFC 3031.

7) Guttman, J. D.: Filtering Postures: Local Enforcement for Global Policies, *Proceedings 1997 IEEE Symposium on Security and Privacy*, IEEE Computer Society, pp. 120–129 (1997).

8) Guttman, J. D., Herzog, A. L. and Thayer, F. J.: Authentication and Confidentiality via IPSEC, *Computer Security - ESORICS 2000, 6th European Symposium on Research in Computer Security, Toulouse, France, October 4-6, 2000, Proceedings*, Lecture Notes in Computer Science, Vol.1895, Sprinter, pp.255–272 (2000).