# FORMALIZATION OF BROADCAST COMMUNICATION IN PROCESS CALCULUS AND ITS MODEL CHECKING

Ritsuya Ikeda, Takuya Ohata and Shin-ya Nishizaki
*Department of Computer Science, Tokyo Insititute of Technology, 2-12-1-W8-69, O-okayama, Meguro, Japan*

Keywords: Automated reasoning, Model checking, Broadcast communication, Process calculus, Pi-calculus, Model checking, Model checker SPIN.

Abstract: A large number of studies have examined communicating processes in formalizing concurrent systems for unicast communications. We propose a process calculus to enable formalizing communicating processes and their computational costs to analyze denial-of-service attack resistance by estimating the cost balance between a victim and attackers. Our system is similar to other process calculi in that it is based on unicast communication. Broadcast communication is also important in the context of denial-of-service attack resistance because several denial-of-service attack methods, such as the Smurf attack, use broadcast communications. Little is known about the formal framework of broadcast communicating processes. In this paper, we formalize broadcast communication in the framework of process calculus and apply it to an analysis of denial-of-service attack resistance of communicating processes via broadcast communication. We propose an extension of the proposed process calculus and an analysis method that uses the SPIN model checker. We give two examples of broadcast communication and verify several properties using the SPIN model checker

## 1 INTRODUCTION

The formalization of communicating processes has been the object of study for a long time. One of the most important frameworks is the pi-calculus (Milner et al., 1992) (Sangiorgi et al. 2004), a system that formalizes communi-cating processes. In that calculus, communication between processes is allowed to be point-to-point or unidirectional; the calculus does not support broad-cast communication. The spi-calculus was proposed to formulate and analyze the security of communication protocols enhanced by adding cryptographic constructs like public-key encryption, shared-key encryption, and hashing (Abadi and Gordon, 1997).

A denial-of-service (DoS) attack is an at-tempt to make a computer service unavailable to its users. The first study of the formalization of DoS attacks on communications protocols and resistance against such attacks was performed by Meadows (2001). She extended the Alice-and-Bob notation by annotating the computational costs in processing data packets. Although the property was deeply related to operational behavior, cost annotation was assigned to each communication operation independently of the operational behavior. We therefore proposed another formal framework called spice calculus; this is based on process calculi where the cost estimation mechanism is linked to operational behavior (Tomioka et al., 2004). We can use this calculus successfully to formalize DoS attack resistance; however, it can only handle point-to-point communication, not broadcast communication.

In this paper, we study the formalization of broadcast communication in spice calculus and a verification method using model checking.

## 2 FORMALIZATION OF BROADCASTING

*Broadcasting* is the transmission of a message to be received by all hosts in a network. It is supported by several network protocols such as Ethernet, token ring, and IPv4. On the other hand, point-to-point transmission is called *unicasting*.

Some DoS attacks on communication protocols use broadcasting as a packet amplifier to overwhelm a victim. The most typical of these is the Smurf attack, in which an attacker sends ICMP echo requests

Ikeda R., Ohata T. and Nishizaki S. (2009).
FORMALIZATION OF BROADCAST COMMUNICATION IN PROCESS CALCULUS AND ITS MODEL CHECKING.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 348-352
DOI: 10.5220/0002276703480352
Copyright © SciTePress

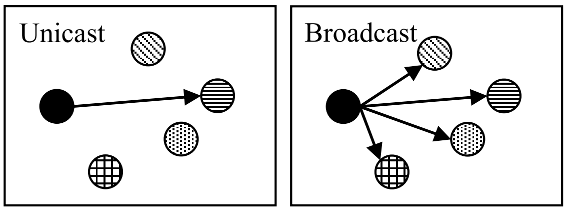

Figure 1: Unicasting and broadcasting.

to IP broadcast addresses whose source IP addresses are spoofed as the victim's IP address (CERT-1998). Each receiver then replies to the victim with an echo reply, and the spoofed echo requests from the attacker are amplified as the echo replies sent from hosts in the network. Broadcasting therefore plays an important role in DoS attacks.

There are two ways of incorporating broadcasting into spice calculus:

1. Implement broadcast communication using unicast communication, which is already provided in the calculus.
2. Introduce broadcast communication primitives into the calculus, just like the unicast primitives, and then define operational semantics of the broadcast primitives.

In this paper, we start with the first option because it works with a model checker.
We implement a process that creates a group of processes simulating a broadcast. We call the process a *broadcast arranger:* the processes in Fig. 3 are generated and arranged as the result of the execution of the process description shown in Fig. 4. We do not explain the description in detail. The code does not implement the processes in Fig. 3 directly, but generates a procedure that creates the processes in Fig. 3. The *relay* process in Fig. 3 shows the "reception" of a broadcast request and the *subrelay* process is a "control clerk" for each host participating the broadcast network. If some host, e.g., HST2, wants to make a broadcast, (1) it sends a message to the channel `broadcast`; (2) (3) then the relay process delivers it to the other subrelays. (4) The subrelays send the message to the corresponding hosts.

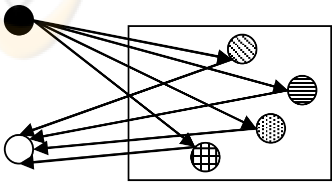

Figure 2: Smurf amplifier.

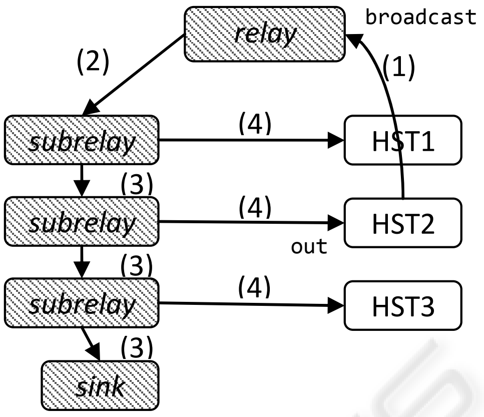

Figure 3: Translated broadcast.

```
new (broadcast);
store last = [];
fork( /* relay arrangement */
 inp con(reply);
 out internal <[broadcast, last]>;
 inp internal (data);
  split [n,out] is data;
  store last=n;
  out reply <[broadcast,out]>;
)
fork(/* sub-relay creation */
  inp internal(data);
  split [b,last] is data;
  new(out); new(n);
  out internal <[n,out]>;
  fork(
    inp n (data);
    out last <data>;
    out out <data>;
  )
)
fork( /* relay creation */
 inp broadcat (data);
 out last <data>
)
```

Figure 4: Description in spice calculus.

```
chan dummy = [0] of { int };
chan broadcast = [0] of { int };
chan con = [0] of { int };
chan last;
byte sink_started = 0;
int node_num = 0;
active proctype sink()
{chan prev = [0] of { int };
 int data;
 atomic {
  last = prev;
  sink_started = 1;
 }
 do ::prev ? data; od
}
active proctype broad_net()
{int data;
 sink_started == 1;
 do ::true ->
   broadcast ? data; last ! data;
 od
}
active proctype add()
{chan reply;
chan prev = [0] of { int };
 chan out;
 chan internal = [0] of { chan, chan };
 sink_started == 1;
 do ::con ? reply ->
   run c(internal);
   internal ! dummy, last;
   internal ? prev, out;
   last = prev; node_num = node_num + 1;
   reply ! out;
 od
}
proctype c(chan internal)
{int data; chan dum; chan next;
 chan prev = [0] of { int };
 chan out = [0] of { int };
 internal ? dum, next;
 internal ! prev, out;
 do ::prev ? data;
    next ! data;
    if ::out ! data; fi;
 od
}
```

Figure 5: Translated code in Promela.

```
% ping -b 192.168.108.255
WARNING: pinging broadcast address
PING 192.168.108.255 (192.168.108.255)
     56(84) bytes of data.
PING 192.168.108.255 (192.168.108.255)
56(84) bytes of data.
64 bytes from 192.168.108.128:
icmp_seq=1 ttl=64 time=0.289 ms
64 bytes from 192.168.108.90:
icmp_seq=1 ttl=64 time=0.479 ms (DUP!)
64 bytes from 192.168.108.65:
icmp_seq=1 ttl=64 time=1.38 ms (DUP!)
64 bytes from 192.168.108.43:
icmp_seq=1 ttl=64 time=1.38 ms (DUP!)
```

Figure 6: Example of Ping.

```
proctype sender(){
 int:data;
 chan pong = [0] of { int };
 broadcast ! ping;
 sended: skip;
 do
  ::pong ? data;
 od
}
```

Figure 7: Ping Sender.

## 3 PRACTICE OF BROADCAST COMMUNICATION IN MODEL CHECKER

Nowadays, many model checkers are implemented and used widely not only in academia but also in industry. Among them, we choose SPIN model checker (Holzmann 2004), which enables us to give a model description in a kind of distributed programming language with synchronous communication via channels. The language is called *Promela* (Process Meta-Language) . The SPIN model checker verifies that a temporal logic's formula satisfies a model written in Promela. We translate a process description of the spice-calculus to a model definition written in Promela. Similarity between the spice-calculus and Promela is multi-process language based on synchronous communication via channel.The difference is process creation: although we can create unlimited number of processes in the spice-calculus, we can use only processes determined by a given program code in Promela. Such limitation is due to efficiency of model checking in SPIN. In translation of the spice-calculus code to Promela's, we naively impose the restriction on process creation: unlimited process creation is translated as fixed number's creation. In Figure 5, the translated code of the spice-calculus to Promela is presented, which is slightly editted for readablility.

### 3.1 Case Study: ICMP's Ping

Ping is a basic network command in several operating systems that enables sending ICMP communications, including ICMP broadcasts. Figure 6 shows an example of using the ping command where a user has broadcast to network 192.168.108.0/24 and four hosts have responded. Code fragments in Figs. 7 and 8 describe the sender of a broadcast and one of four responders, respectively. The sender sends a broadcast packet *ping* and receives *pong* responses. The responder sends a *pong* back. The two code fragments provide a model for the SPIN model checker.

```
proctype sender(){
 int:data;
 chan pong = [0] of { int };
 broadcast ! ping;
 sent: skip;
 do
  ::pong ? data;
 od
}
```

Figure 8: Ping sender.

```
proctype responder1(){
 chan reply = [0] of { chan };
 chan out = of { int };
 chan pong; int prefix = 1; int data;
 con ! reply;
 reply ? out;
 do
  ::out ? pong -> ping ! prefix;
    received: skip;
 od
```

Figure 9: Ping Responder.

An assertion to be checked by SPIN is described in linear temporal logic as

```
[](sender@sent ->
  (<>responder1@received &&
   <>responder2@received &&
   <>responder3@received)),
```

where `sender@sent` means reachability in line 5 of Fig. 8, and `responder1@received` means reachability at line 9 of Fig. 9. The formula denotes that if control reaches `sent` in process `sender`, then it will finally reach the `received` points in `responder1`, `responder2`, and `responder3`. The formula was successfully checked by the SPIN model checker.

## 3.2 Case Study: Smurf Attack

We next formalize the Smurf DoS attack using our framework. In a Smurf attack, an attacker transmits a spoofed broadcast message with the victim's address to a network (CERT 1998). Those receiving the broadcast send messages back to the victim, and a large number of messages arrive at the victim to cause DoS.

```
byte victim_ready = 0;
proctype attacker(){
 int ping_num = 0;
 victim_ready == 1;
 broadcast ! victim_chan;
 ping_num++;
 smurf_attacked:skip;
}
```

Figure 10: Smurf attacker.

In the example used in this section, we assume one attacker, one victim, and three broadcast listeners.

Figure 10 represents a Smurf attacker and Fig. 10, a victim. We assumed three other receivers of the attacker's broadcast as shown in Fig. 11. An assertion to be checked by SPIN is given as a never claim, which describes an automaton that should not terminate in the final state. Figure 12 represents "it holds globally that if the attacker sends a ping, then the victim will finally receive more than four packets in total."
As shown in Fig. 13, this was successfully checked by the SPIN model checker, showing the property of leverage, which causes the DoS effect in the Smurf attack.

## 4 CONCLUSIONS

We have proposed a method of formalizing broadcast communication in a process calculus and of verifying formal properties using the SPIN model checker. We presented two case studies: the ICMP ping and the Smurf attack. Although these two cases are quite rudimentary, they demonstrate the merit of our model-checking method compared to other methods such as simulation.

At first, we did not check the condition in line 15 of Fig. 5, "`sink_started == 1;`". Without this condi-tion, if the sink process stops, then whole the system stops, causing an error. First, we tried to find the error using the random simulation mode of SPIN rather than comprehensive model checking, but we were unsuccessful. We then successfully found the error using comprehensive model checking. Several issues remain to be addressed.
First, we should study broadcast communica-tions more formally. In our work, we encoded the broadcast in the spice calculus and Promela; al-though this approach seems reasonable, it is not yet theoretically

```
int victim_ping_received = 0;
prototype victim(){
 chan reply = [0] of { chan };
 chan out = [0] of { int };
 chan pong; int prefix = 0; int data;
 con ! reply;
 reply ? out;
 node_num >= 4;
 victim_chan = out; victim_read = 1;
 do :: out ? pong ->
         victim_ping_received++;
 od;
}
```

Figure 11: Smurf victim.

justified. We should axiomatize broad-cast communication in some concurrent theory to justify our encoding.

Second, we provided two examples that are very rudimentary. DoS attacks occur in other environments such as internet routing algorithms. We should try to formalize more practical examples.

```
#define p
  (smurf_attacker@smurf_attacked)
#define q (victim_ping_received >= 4)

never {    /* !([](p -> <>q)) */
T0_init:
      if
      :: (! ((q)) && (p)) ->
               goto accept_S4
      :: (1) -> goto T0_init
      fi;
accept_S4:
      if
      :: (! ((q))) ->
              goto accept_S4
      fi;
}
```

Figure 12: Never claim.

## REFERENCES

Abadi, M., Gordon, A. D., 1997. A Calculus for Cryptographic Protocols: The Spi Calculus. In *Fourth ACM Conference on Computer and Communications Security,* pp. 36-47. ACM Press.

Milner, R., Parrow, J., Walker, D., 1992. A Calculus for Mobile Processes, Part I and Part II. *Information and Computation,* Vol. 100, No. 1, pp. 1-77.

Sangiorgi, D., Walker, D., 2004. The Pi-Calculus: A Theory of Mobile Processes, Cambridge University Press.

Meadows, C., 2001. A Cost-Based Framework for Analysis of Denial-of-Service in Networks. *Journal of Computer Security,* Vol. 9, No. 1/2, pp. 143-164.

Tomioka, D., Nishizaki, S., Ikeda, R., 2004. A Cost Estimation Calculus for Analyzing the Resistance to Denial-of-Service Attack. In *Software Security – Theories and Systems,* Lecture Notes in Computer Science*,* Vol. 3233, Springer, pp. 25-44.

Holzmann, G. J., 2004. The Spin Model Checker – Primer and Reference Manual. Addison-Wesley.

CERT 1998. Smurf IP Denial-of-Service Attacks, CERT Advisory CA-1998-01,
http://www.cert.org/advisories/CA-1998-01.html