Final Report: Modeling the SIP Proxy using Promela

Jong Yul Kim (jk2520)
December 23, 2009

I. Overview

The Session Initiation Protocol (SIP) [1] is a signaling protocol used to set up and tear down multimedia sessions between two endpoints over the Internet. The endpoints are called user agents: user agent client (UAC) initiates a new session to a user agent server (UAS) by sending an INVITE request. The UAS responds with zero or more provisional response such as "180 Ringing" and one final response such as "200 OK" or "486 Busy Here". There are also intermediaries that route requests for user agents called proxy servers. Figure 1 shows the basic setup of two user agents and two proxy servers in an arrangement known as the "SIP trapezoid".

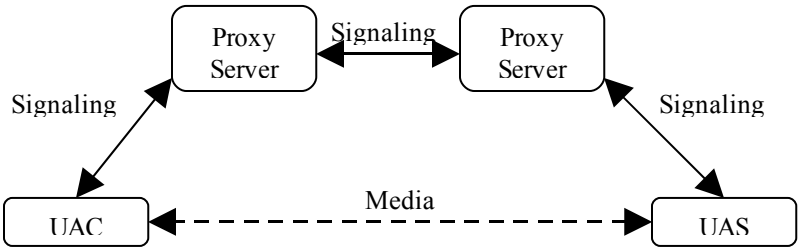

Figure 1. The basic SIP trapezoid

There are two types of proxy servers in SIP: stateless and stateful. A stateless proxy is one that simply forwards a SIP request downstream based on the information already provided within the request. Once the request leaves the proxy, the transaction is forgotten. On the other hand, a stateful proxy can do many more things such as performing sequential or parallel "ring" of the recipient's devices. Trying the office phone first then trying the cell phone is an example of a sequential search. Ringing all phones simultaneously is an example of a parallel search. The SIP proxy is described in detail in RFC 3261 [1].

The purpose of this project is to build a model of the interactions between user agents and a SIP proxy, both stateless and stateful. It was inspired by work done at AT&T labs where a formal state-based model of a user agent client and a user agent server was used to verify the session

establishment protocol for correctness and completeness [2]. Problems with the protocol, such as flows with undefined behavior and race conditions, were discovered in the process of building the model. The project reuses and extends the user agent models that came out of this work and also aims to check the protocol involving proxies for correctness and completeness.

The contribution of this project is the following.

- Promela models of
  - a stateless proxy
  - a stateful proxy that performs sequential forwarding of an INVITE request
  - a UAC that can handle two early dialogs
- Verification of the models using the SPIN model checker

The rest of the report is organized as follows: Section II describes the AT&T work and its limitations in more detail, Section III describes the limitations of the models, Section IV and V describes the stateless and stateful models respectively, Section VI explains the relationship of this project to the topics discussed in class, and Section VII concludes the report with some future work ideas.

II. The UAC and UAS models

Since the project builds on another model, the model is described briefly along with its effects on the project. The UAC and UAS models built at AT&T labs focuses on dialog creation initiated by an INVITE request and the subsequent media negotiations between two user agents. A total of seven requests (INVITE, PRACK, ACK, INFO, UPDATE, CANCEL, and BYE) involved in session setup and termination, and their corresponding success/failure responses are represented.

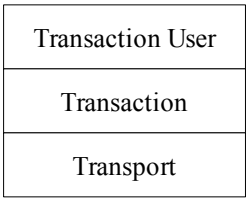

Figure 2: Layers of the SIP stack

The models specify the transaction user (TU) layer of the SIP stack as shown in Figure 2. The TU layer is where new transactions are initiated and responded to. It is also called the UAC core, UAS core, or proxy core depending on which SIP entity is being described because it is in this layer that the high level logic is reasoned and implemented. Therefore, the models safely assume that any request or response sent over the network is syntactically well-formed and guaranteed to arrive at the correct destination without packet loss. Complications that arise due to lower layer behaviors such as retransmissions, timeouts, and message syntax verifications are left out of the models.

Three different types of models are presented in [2]. The basic model captures unordered delivery of messages, the FIFO model assumes ordered delivery of messages by using one TCP connection each from UAC and UAS, and the pruned model reduces some complexities from the FIFO model by eliminating redundant message types from the UAC and UAS. The project uses the pruned model[1] as it is the least complex of the three models, as shown in Figure 3. Using the

1 The pruned model was not available online so I modified the FIFO model manually based on the explanations in [2]. The reconstructed pruned model has been verified before being used in the project.

model with the least complexity is very important because adding another entity, i.e. the proxy, to the model will invariably result in state space explosion.

Performance Measure	Basic Model	FIFO Model	Pruned Model
lines of reachable code	404	300	266
state vector (bytes)	200	88	88
depth reached	6,165	1,068	464
state transitions	780,538,240	9,043,855	3,429,348
memory usage (Mbytes)	20,904	308	105
elapsed time (seconds)	4,200	38	13

Figure 3: Complexity of the three models

There are other simplifying assumptions that the models make. Different messages of the same type are not distinguished and therefore behavior that depend on message fields are not represented. The models do not distinguish between various failure responses that may make the UAC behave in different ways. Also, the models assume a one-to-one session setup between one UAC and one UAS. All of these assumptions affected the scope of the project and the last one in particular presented a challenge that was solved by modifying the UAC model.

III. Limitations

Utilizing an explicit state model checker like the SPIN model checker to verify a non-trivial model usually implies using a machine with lots of resources or somehow trying to keep the state space small so that all states can be verified. The project takes the latter approach.

The project builds on the pruned model of UAC and UAS from the previous work. The pruned model is the least complex of the three models and also has the most limited coverage of the SIP protocol. For example, some valid messages flows are left out of the models in order to reduce complexity and the UAC and UAS use two TCP connections to communicate throughout the session so that the messages are delivered in order. The same limitations carry over to the project by having UAC and UAS connect to the proxy with two TCP connections as shown in Figure 4.

Furthermore, the models in this project contain only one proxy server and one UAC and limits the maximum number of UAS to two. Verifying multiple proxies or more than two UAS is not possible.

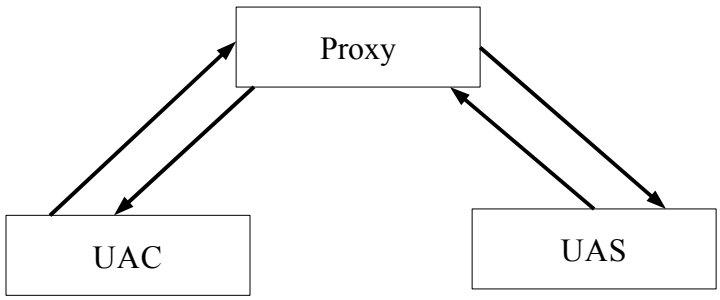

Figure 4: Model of one proxy, one UAC, and one UAS

Lastly, some proxy behavior is not represented in the model. For example, when a stateful proxy forwards INVITE requests to multiple UAS, the proxy has to distinguish various failure responses. The proxy in the real world reacts differently to a global failure as opposed to a temporary failure but in the model there is no such distinction. Every failure is considered a temporary failure.

IV. Model of a stateless SIP proxy

As explained earlier, a stateless proxy "blindly" forwards messages from one end to the other. The proxy model in this case is very simple and the code is shown in Figure 5. c2p is the channel from UAC to proxy, p2s is the channel from proxy to UAS, s2p is the channel from UAS to proxy, and p2c is the channel from proxy to UAC. The proxy basically relays message from UAC to UAS and from UAS to UAC.

```
proctype proxy() {
  mtype message, sdp;
end:   do
       :: c2p?message,sdp; p2s!message,sdp
       :: s2p?message,sdp; p2c!message,sdp
       od
}
```

Figure 5: Promela code for stateless proxy

```
proctype proxy() {
  mtype message, sdp;
  byte uasid;
       c2p?invite,sdp;
       do
       :: true; uasid = 0; break;
       :: true; uasid = 1; break;
       od;
       p2s[uasid]!invite,sdp;
end:     do
       :: c2p?message,sdp; p2s[uasid]!message,sdp
       :: s2p[uasid]?message,sdp; p2c!message,sdp
       od
}
```

Figure 6: Promela code for stateless proxy with two UAS

When there are multiple candidate UAS, then it picks one and forwards the message consistently to the one it picked. The consistent forwarding behavior comes not from keeping state, but from deterministically resolving the destination using information in the message itself. Figure 6 shows the Promela code for this model. The proxy picks either UAS 0 or UAS 1, stores the value in a variable called *uasid*, and then sends all subsequent messages to that UAS. Of course, in real world implementation, there is no such variable as *uasid* since that would be like keeping state. The variable represents not state but the deterministic resolution of destination of stateless proxies. The UAS was only slightly modified so that two instances can run instead of one.

Verification was successful in that there were no assertion violations and no invalid end states. The results are shown in the table below.

	Stateless proxy with one UAS	Stateless proxy with two UAS
State vector (bytes)	84	84
Depth reached	890	890
State transitions	136,623,940	136,623,940
Memory usage (Mbytes)	1,908	1,908
Time elapsed (seconds)	446	1,430

V. Model of a stateful SIP proxy

A stateful proxy is a proxy that keeps track of transaction states of the interaction between UAC and UAS so that it can provide more interesting features. In RFC 3261, it is modeled as having one server transaction unit that faces the UAC and one client transaction unit that faces the UAS. If there are multiple candidate UAS, then there are matching number of client transaction units within the proxy.

Modeling the full proxy as described in the RFC would incur exponential state space, so the project (once again) takes the approach of simplifying the model. The model focuses on sequential search with two UAS where the proxy tries to contact one candidate UAS, and if it fails, then tries to contact the other UAS. If at least one of the UAS returns a success response, then the end-to-end call goes through. Otherwise, the call fails.

The Promela code for this proxy is shown in Figure 8. The proxy sends an INVITE to one UAS first, and if it gets an *invFail* response, then it will forward it to the other UAS. If any succeeds, then the proxy relays messages between the succeeding UAC and UAS pair. If both fail, then a single final *invFail* response is sent to the UAC as specified in RFC 3261.

RFC 3261 also states that the proxy must relay any provisional responses and success responses to the UAC immediately. This posed problems in the model because the pruned model of UAC didn't account for the fact that there may be multiple early dialogs that are setup by these provisional responses from multiple UAS. Therefore, a significant amount modification was done to the UAC so that it would handle multiple early dialogs correctly. The resulting code is long and somewhat ugly but has been verified to the extent possible.

Even with all the simplifications, the verification of the modified UAC and the stateful proxy had to be performed with supertrace/bitstate search and hash-compact search because SPIN ran out of memory with exhaustive search in a shared machine with 4 GB of physical memory. The main reason is probably the modifications done to UAC to handle multiple early dialogs. Each variable changed to an array would have led to state space explosion. The extra logic implemented in the proxy would have an effect as well.

```
proctype proxy() {
  mtype message, sdp, uacsdp;
  byte uasid = 0;
  bool reinvite[2] = false;
       c2p?invite,uacsdp,uasid;
       p2s[0]!invite,uacsdp;
end1:    do
       :: c2p?message,sdp,uasid; assert(uasid==0); p2s[0]!message,sdp;
          if :: message == invite; reinvite[0] = true; :: else; fi;
       :: s2p[0]?message,sdp;
          if
          :: reinvite[0] == false && message == invFail; goto nextUAS;
          :: else; p2c!message,sdp,0;
          fi;
       od;
nextUAS:  p2s[1]!invite,uacsdp;
end2:    do
       :: c2p?message,sdp,0; p2s[0]!message,sdp;
       :: c2p?message,sdp,1; p2s[1]!message,sdp;
          if :: message == invite; reinvite[1] = true; :: else; fi;
       :: s2p[1]?message,sdp;
          if
          :: reinvite[1] == false && message == invFail; goto fail;
          :: else; p2c!message,sdp,1;
          fi;
       od;
fail:    p2c!invFail,none,0;
endclean:  do
       :: c2p?message,sdp,0; p2s[0]!message,sdp;
       :: c2p?message,sdp,1; p2s[1]!message,sdp;
       :: s2p[0]?message,sdp; p2c!message,sdp,0;
       :: s2p[1]?message,sdp; p2c!message,sdp,1;
       od;
}
```

Figure 8: Promela code for statelful proxy that forwards an INVITE to two UAS sequentially

The result of verifying this model is summarized in the table below.

	With supertrace	With hash-compact
State vector (bytes)	172	172
Depth reached	300	1,442
State transitions	32,126,542	137,482,750
Memory usage (Mbytes)	17	1,808
Time elapsed (seconds)	81	9,660

VI. Relationship to materials covered in class

This project uses Promela and the SPIN model checker, which performs explicit state model checking. The model is in the form of a Kripke structure and properties can be verified using LTL. Therefore, it uses materials from the first five lectures in class.

However, it does not use any material from symbolic model checking, hardware-related model checkers, and SAT solvers.

VII. Conclusion

SIP is a complex protocol that utilizes many different and sometimes redundant messages to setup and teardown a multimedia session. Trying to verify the correctness and completeness of a certain aspect of the protocol using an explicit state model checker like SPIN requires the careful use of abstraction and simplification.

In this project, an attempt was made to verify the interactions of user agents and a proxy and to find any lapse in completeness or correctness in the protocol by applying such abstraction and simplification. There were no protocol mistakes found, but the process of making the models has helped in gaining a better understanding of the usefulness and limitations of a model checking tool, and also in gaining a better understanding of a small part of the SIP protocol.

Due to time constraints, further simplifications couldn't be applied to the UAC. The UAC code contains a lot of message unnecessary to the understanding of stateful proxy behavior. For example, in the viewpoint of the proxy, an INFO message or a UPDATE message from the UAC is simply a message that needs to be forwarded to the UAS once an early dialog is established. These complexities could have been eliminated. This is left as future work.

VIII. References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E. "SIP: Session Initiation Protocol" IETF Network Working Group Request for Comments 3261, 2002.

[2] Zave, Pamela. "Understanding SIP through Model-Checking" IPTComm 2008, LNCS 5310, pp. 256-279, 2008.