基于时间着色 Petri 网的 SIP 协议验证与分析

刘 靖 叶新铭 马元飞
(内蒙古大学计算机学院 呼和浩特 010021)

摘 要 随着 SIP(Session Initiation Protocol)被 3G 通信选择为下一代移动网络的会话控制机制,保证 SIP 协议设计和实现无缺陷、运行稳定可靠成为 SIP 协议应用过程中亟需研究和解决的关键问题。充分利用时间着色 Petri 网(Timed Colored Petri Nets,TCPN)在描述和分析具有复杂交互行为及时间约束的系统方面的优势,给出了 SIP 协议的层次 TCPN 模型,并集成多种模型分析技术,完成 SIP 协议设计的正确性验证;同时通过正则表达式完成协议模型的生成路径分析,指出其中存在的死锁状态并分析原因。提出了相应的协议设计改进方案,验证了设计方案的正确性,从而有效增强了 SIP 协议在实际应用中的可行性和可靠性。

关键词 SIP,时间着色 Petri 网,协议验证,死锁分析
中图法分类号 TP319 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.07.025

Verification and Analysis of SIP Protocol Based on Timed Colored Petri Nets

LIU Jing YE Xin-ming MA Yuan-fei
(College of Computer Science, Inner Mongolia University, Hohhot 010021, China)

Abstract SIP(Session Initiation Protocol) has been selected by 3G communication as a session control mechanism for the next generation mobile network, so it is quite significant to ensure that the protocol design and implementation is defect-free and runs steadily and reliably. Timed Colored Petri Nets(TCPN) has advantages of modeling and analyzing software systems with complicated and time-constrained behaviors. Thus, TCPN was well adopted in this paper to construct a hierarchical formal model for SIP protocol, and several model analysis techniques were used together to validate its design accuracy. Then, using regular expression, the model based protocol execution paths were completely analyzed, and certain deadlock scenarios were pointed out. Finally, we proposed novel and validated protocol design revisions to effectively improve the feasibility and the reliability for practical SIP applications.

Keywords SIP, Timed colored Petri nets, Protocol verification, Deadlock analysis

1 引言

SIP(Session Initiation Protocol)协议[1,2]以无缝、灵活、可扩展特性被 3GPP(The 3rd Generation Partnership Project)确定为第三代移动通信系统的 IMS(IP Multimedia Concepts and Services)信令协议,并将逐步成为 NGN(Next Generation Network)中的核心控制协议之一。它能够满足 VoIP 和多媒体通信方面的要求,因而许多通讯公司都相继推出了支持 SIP 的服务和终端产品。为了适应上述技术和应用发展,需要对 SIP 协议进行必要的扩展和完善,但协议设计中的任何错误和缺陷都将给系统的稳定性、可靠性带来巨大影响,因此为了保证产品的可用性和一致性,降低后期的测试和维护成本,如何在应用开发早期以较低的代价发现协议中潜在的问题并提出改进方案是一个值得深入且持续研究的关键性问题。

采用形式化方法对协议进行建模以及分析,可以有效确认协议设计的正确性和可用性。本文采用时间着色 Petri 网(Timed Coloured Petri Nets,TCPN)[3]作为基础的形式描述和分析模型,充分利用其在描述具有复杂交互行为及时间约束的系统行为、动态模拟系统执行、分析验证系统属性等方面的优势,为 SIP 协议构建层次 TCPN 模型,并集成多种模型分析技术,实现协议的设计验证和缺陷确认。

目前对 SIP 协议的主要研究集中在 3 个方面:SIP 测试方法与工具研究[4];基于 SIP 的各种服务及应用[5,6];SIP 与其他协议的协同工作[7,8]。对 SIP 协议设计的形式化分析研究较少,典型工作包括:文献[9,10]采用基本的着色 Petri 网对 SIP 协议进行验证,发现协议存在的死锁状态,并提出通过添加定时器来解决死锁的改进思路,但是对核心的定时器触发时机并未给出具体解决方案。而本文采用时间着色 Petri 网给出描述更为准确、更利于分析的协议模型,并给出更为全面可行的改进方案。文献[11]采用着色 Petri 网对 SIP 的 non-INVITE 事务进行了验证,但模型较为庞大,缺乏清晰的层次结构,没有对协议的时间约束条件进行验证。文献[12]采用广义随机 Petri 网对 SIP 进行了建模与性能分析,但对协

到稿日期:2013-04-30 返修日期:2013-05-30 本文受国家自然科学基金项目(61262017),高校博士学科点专项科研基金(20101501110003),内蒙古自然科学基金重点项目(20080404Zd20),内蒙古大学高层次人才引进基金项目资助。

刘 靖(1981—),男,博士,讲师,主要研究方向为网络协议验证与测试、Petri 网理论与应用,E-mail:liujing@imu.edu.cn;叶新铭(1943—),男,教授,博士生导师,主要研究方向为计算机网络、形式化方法;马元飞(1985—),男,硕士,主要研究方向为网络协议验证。

议的交互细节进行了高度抽象建模，模型和验证过程都比较简单。

针对上述的问题和已有研究的不足，本文采用 TCPN 作为基础的形式描述和分析模型，构建了 SIP 协议的层次 TCPN 模型，实现不同状态下协议功能行为的分层抽象建模，同时集成多种模型分析技术，确认模型描述准确并完成协议设计正确性的验证。此外，我们通过正则表达式完成协议模型的生成路径分析，指出协议存在的死锁状态，继而提出了相应的协议设计改进方案，并验证了改进方案的正确性，有效增强了 SIP 协议在实际应用中的可行性和可靠性。

2 SIP 协议流程

SIP 协议主要通过一系列的事务(Transaction)来完成对会话(session)的控制，INVITE 事务与 non-INVITE 事务是两个主要的 SIP 事务。INVITE 事务的主要功能是建立一个会话，non-INVITE 事务的功能是维持、关闭一个会话。SIP 端系统称为 UA(User Agent)，指为了向服务器发送请求而与服务器建立连接的应用程序。UA 通过与服务器之间发送请求和接收响应来完成呼叫和传送层的控制。因为 UA 既要发出呼叫，又要能接受呼叫，所以一个 UA 包括一个用户代理服务器(User Agent Server，UAS)和一个用户代理客户端(User Agent Client，UAC)。作为 SIP 协议的两个关键参与者，UAC 是一个逻辑实体，负责创建请求；然后使用事务状态机描述的变化规则来传送请求，UAS 是对 UAC 所发出的请求进行应答的逻辑实体。

SIP 中的请求消息包括邀请(INVITE)、确认(ACK)、可选项(OPTIONS)、再见(BYE)、取消(CANCEL)和注册(REGISTER)等 6 种。同样，SIP 核心协议定义了临时响应(1xx)、成功响应(2xx)、重定向响应(3xx)、客户机错误(4xx)、服务器错误(5xx)、全局故障(6xx)等 6 种类型的响应消息。SIP 支持 3 种呼叫方式：由 UAC 向 UAS 直接呼叫、由 UAC 在重定向服务器的辅助下进行重定向呼叫、由代理服务器代表 UAC 向被叫发起呼叫。本文重点对具有基础性作用且应用最多的直接呼叫方式进行建模分析。图 1 描述了 UAC 向 UAS 直接呼叫的建立过程，交互流程的具体细节可参照文献[1]。

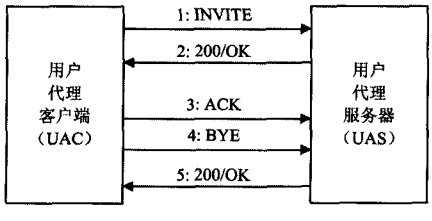

图 1 UAC 向 UAS 直接呼叫的建立过程

3 SIP 协议的 TCPN 建模

我们采用 CPN Tools[13] 对 SIP 协议中的 INVITE 事务进行分层建模，基于不同的模型抽象层次对协议行为加以描述并细化模型，这样既可以合理约束协议建模范围，辅助协议功能的组织，将协议细节行为集中描述在一个层次中，便于对协议模型的局部完善和改进；又能够有效控制模型大小，易于进行协议模型的确认与协议性质的分析。

如表 1 所列，SIP 协议的整体 TCPN 模型包括 10 个无重复的页(page)，位于不同的建模层次，每页模型描述了各自抽象级别的协议功能，抽象级别低的模型页作为抽象级别高的的模型页的替代变迁的子页实现。其中，Server 页用来描述 UAS 行为，Client 页用来描述 UAC 行为，Net 页用来模拟不可靠链路行为。根据 UAC 和 UAS 在 INVITE 事务执行中所具有的不同状态，完成各自层内模块的细化，其中并未对 terminated 状态进行独立描述，因为在该状态下协议不具有任何行为，其只是用来表示本次事务的终止。

表 1 SIP 协议 TCPN 模型中的页层次描述

模型页	所建模功能的描述
Top	协议总体运行架构及网络拓扑
Client	UAC 行为的总体执行流程
Ccalling	UAC 在 calling 状态的具体行为
Cproceeding	UAC 在 proceeding 状态的具体行为
Ccompleted	UAC 在 completed 状态的具体行为
Server	UAS 行为的总体执行流程
Sproceeding	UAS 在 proceeding 状态的具体行为
Scompleted	UAS 在 completed 状态的具体行为
Sconfirmed	UAS 在 confirmed 状态的具体行为
NET	网络传输行为

3.1 关键数据建模

表 2 列出了 SIP 协议 TCPN 模型中的关键数据描述，主要包括如下几个部分。

表 2 SIP 协议 TCPN 模型中的关键数据描述

关键数据的 colset 定义
colset REQUEST=with INVITE | ACK | NOREQU timed
colset RESPONSES=with r101|r100|r2xx|r3xx|NORESP timed
colset MESSAGE=with TA|TB|TD|TG|TH|TI|T200|TRR|RMFU|RMFD|NOMESS timed
colset STATEC=with INI|CALL|PROC|COMP|COMPR|TERM timed
colset STATES=with SINI|SPROC|SCOMP|SCOMPR|SCONF|SCONFR|STERM timed
colset SCENEC = product MESSAGE * REQUEST * RESPONSES * STATEC * STATEC timed
colset SCENES = product MESSAGE * RESPONSES * REQUEST * STATES * STATES timed
colset LSCENEC=list SCENEC
colset LSCENES=list SCENES

• REQUEST 表示 UAC 可以发送的两类消息 INVITE、ACK，NOREQU 表示没有任何消息发送。RESPONSE 表示 UAS 发送的应答消息类型，有 r101，r100，r2xx，r3xx 4 种类型。在 SIP 协议的 RFC 中，r3xx 到 r6xx 的处理方式一致，所以不失一般性可以用 r3xx 表示从 r3xx 到 r6xx 类型的消息。

• MESSAGE 表示 UAC 和 UAS 端所发生的事件消息，其中 TA、TB、TD 分别表示 UAC 端定时器 A、B、D 的触发，TG、TH、TI、T200 分别表示 UAS 端定时器 G、H、I 以及 200ms 内服务器无应答时的定时器触发。RMFD 与 RMFU 用来模拟与相邻层信息的交互，RMFU 表示从协议上层模块收到数据的事件，RMFD 表示从协议下层模块收到数据的事件。NOMESS 表示没有事件发生。

• STATEC 和 STATES 分别表示 UAC 和 UAS 端在交互过程中经历的不同状态。以 UAC 为例，其中添加 INI 状态表示 UAC 未收到上层实体所发送的消息时所处的状态，即为了模拟 INVITE FROM TU 前所添加的初始态。COMP 与 COMPR 都表示 UAC 处在 Completed 状态，区别在于：

UAC 从 Proceedings 首先转到 COMP 状态，在 COMP 状态下收到 3xx－6xx 应答后转到 COMPR 状态，在 COMPR 状态下允许对传输层错误进行处理，这样，UAC 在 Completed 状态下至少可以收到一次应答，从而避免了不可靠链路中传输层错误频繁发生对 UAC 和 UAS 交互结果的影响，防止 UAC 总是因传输层错误而结束交互。

• SCENEC 和 SCENES 分别用一个五元组叉乘(product)组合类型来描述 UAC 和 UAS 的行为，其中元素 MESSAGE 表示发生的事件消息，第 2 个和第 3 个元素分别表示该事件引发的行为和触发该事件的行为。第 4 个和第 5 个元素分别表示发生该事件前后所处的不同状态。这样描述协议行为的方式一方面便于在模拟过程中收集协议运行状态的数据，另一方面也便于使用正则表达式对协议行为执行路径进行分析处理。

• LSCENEC 和 LSCENES 分别以列表类型(list)存储 UAC 和 UAS 端生命周期中行为的各次变化情况。

特别说明，上述数据描述中，涉及时间因素的数据都在其定义后加上了 timed 标识，记录该类型数据托肯(token)的可用时间，并根据模型全局时间来判定并触发相关变迁的点火执行。在 INVITE 事务建模中，UAC 与 UAS 的交互过程共涉及到 6 个定时器，分别是 UAC 端的 timerA、timerB 和 TimerD，以及 UAS 端的 TimerG、TimerH 和 TimerI。以 timerA为例来说明定时器事件的建模。当 UAC 处在 Calling 状态时，如果过了 $T1$ 时间间隔后($T1$ 表示基本往返时延 RTT，在 SIP 的协议规范中可置为 500ms)仍未收到 UAS 端的应答，则 timerA 触发，导致 INVITE 请求重发。同时，将 timerA 的触发间隔设置为 $2 * T1$。若过了 $2 * T1$ 时间间隔仍未收到 UAS 端的应答，则再次触发 timerA，导致 INVITE 请求的再次重发，同时，将 timerA 的触发间隔设置为 $4 * T1$。以此类推，直到 timerA 第 6 次触发，客户端的 INVITE 请求会最后重发一次。当时间间隔为 $64 * T1$ 时，如果还未收到服务器端的应答，则 timerB 触发。其余定时器的建模都可以以 $T1$ 为基础，将 $T1$ 用在弧表达式以及库所初值中，通过变迁的触发来模拟定时器的定时规律。

3.2 总体行为流程建模

SIP 协议 TCPN 模型的顶级页(top page)如图 2 所示，它描述了协议运行的总体架构及网络拓扑。图中用 3 个替代变迁分别描述了 UAC、UAS 和网络(NET)在协议交互过程中的通信交互，即 UAC 将 REQUEST 类型数据通过 NET 发送给 UAS，而 UAS 通过 NET 向 UAC 回传 RESPONSES 类型数据。

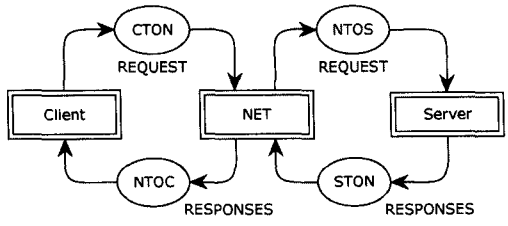

图 2 SIP 协议 TCPN 模型的 Top 页

作为示例，图 3 给出图 2 中替代变迁 Client 对应的页模型，用它来描述 UAC 端的行为流程。其中 3 个替代变迁所对应的子页模型可以更具体地描述在不同状态下 UAC 端的具体行为。通过变迁 TransErr 的防卫表达式以及 trans_err(st)函数来模拟协议处在 Calling、Completed 等不同状态下发生传输层错误的处理方式。库所 Scene 建模为融合集，统一描述 UAC 的行为变化。

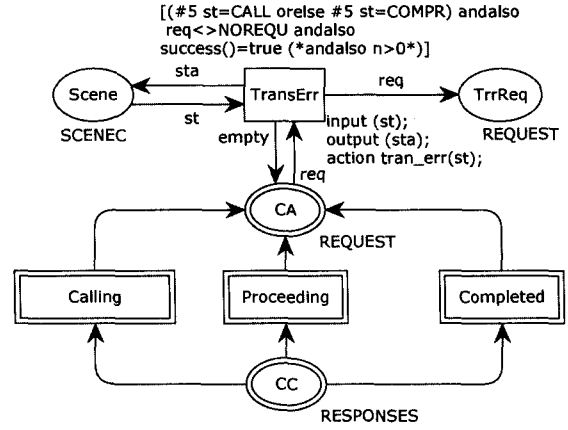

图 3 SIP 协议 TCPN 模型的 Client 页

3.3 网络环境建模

作为示例，图 4 给出的 NET 页模型描述了从 UAC 端到 UAS 端的网络传输功能建模。其中，库所 Schannel_Em 记录成功传送到 UAS 端的消息个数；库所 CollectorCTS 用来收集不可靠链路丢失的消息。变迁 CTOS 与 RCTS 用来模拟不可靠链路，具体而言，变迁 CTOS 防卫表达式[b=successnu()]中的函数 successnu 定义为 uniform(0.0,1.0)<0.5，对布尔型变量 b 进行赋值，则 b 取真值的概率为 50%；以从 CTOS 到 NB 的弧表达式为例，if b=false then empty else 1`req，根据 b 的取值决定是否发送消息，由于 b 取真的概率为 50%，因此消息成功到达服务器的的概率为 50%。同理，变迁 RCTS 用来模拟不可靠链路上的重复数据包，并对数据包到达时延进行模拟，即变迁执行时间设置为@+1，表示每个重复数据包延迟 1 个时间单位到达。如此，通过防卫表达式、弧表达式以及时间类型的使用，可以对存在丢包、延迟、重复数据包的不可靠链路进行模拟，而且通过更改其中一些参数，可以动态地调整链路的可靠程度，从而达到真实模拟不可靠链路的要求。

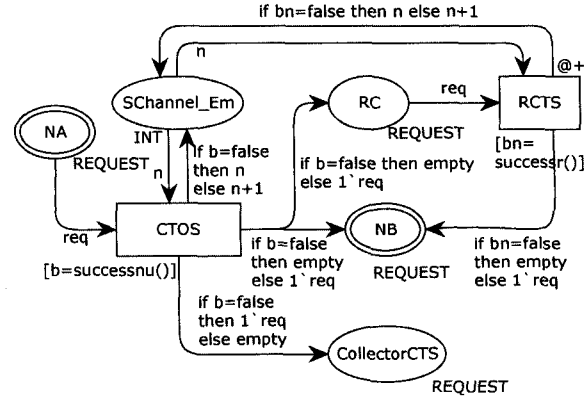

图 4 SIP 协议 TCPN 模型的 NET 页

3.4 具体协议功能建模

表 1 中 Ccalling、Sproceeding 等底层模型页描述 UAC 端和 UAS 端在不同状态下对消息和事件的具体处理过程，即对协议的具体功能行为进行建模。本节以 UAC 端在 Calling 状

态下的行为建模为例，说明具体功能的模型描述方式，重点包括定时器触发事件、应答消息处理等两个核心方面。

图 5 给出 UAC 端在 Calling 状态下对 INVITE 消息处理行为的模型页，是图 3 中替代变迁 CCalling 所对应的具体功能子页。其中，变迁 CallTimer 描述 UAC 端在超时状态下的消息处理行为，变迁 CallResp 描述 UAC 端在收到 UAS 端请求应答时的消息处理行为。库所 TimerAorB 用来控制定时器 A 和 B 的触发，融合库所 cloneCs 用队列收集每次 UAC 端的状态变化，其中队首元素表示 UAC 当前所处的状态，融合库所 Scenec 记录 UAC 当前所处的状态，以及导致到达该状态所发生的事件。

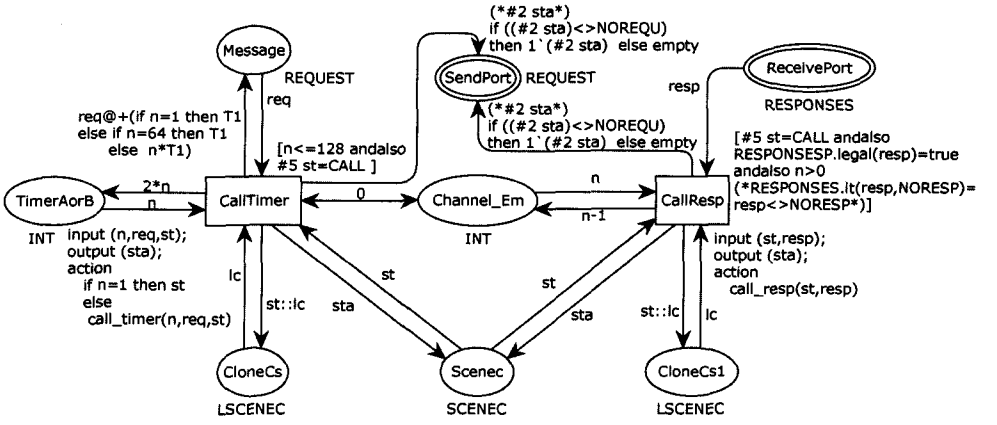

图 5　SIP 协议 TCPN 模型的 CCalling 页

• 定时器触发事件的建模

当 TimerA 或 TimerB 超时发生时，变迁 CallTimer 点火执行，其附属的代码段也将同时执行。该代码段中 st 与 sta 都是 SCENEC 类型的变量，st 为超时事件发生前的 UAC 状态，sta 为经过超时处理后的 UAC 状态；判定条件 $n=1$ 标识未发生超时且 st 不变，否则直接调用 call_timer(n,req,st)函数完成对超时事件的处理，该函数具体描述如下：

```
fun call_timer(n:INT,req:REQUEST,st:SCENEC) =
  if n<=64 then//表示发生的是 TimerA 超时
    let
      val a=SCENEC.set_1 st TA;
      val b=SCENEC.set_2 a req;
      val c=SCENEC.set_3 b NORESP;
      val d=SCENEC.set_4 c (#5 st);
    in
      SCENEC.set_5 d CALL;
    End
  else//表示发生的是 TimerB 超时
    let
      val a=SCENEC.set_1 st TB;
      val b=SCENEC.set_2 a NOREQU;
      val c=SCENEC.set_3 b NORESP;
      val d=SCENEC.set_4 c (#5 st);
    in
      SCENEC.set_5 d TERM;
    end
```

由上可知，UAC 发生 TimerA 超时后所处的状态为 CALL，而发生 TimerB 超时后所处的状态为 TERM。

• 应答消息处理的建模

当收到来自 UAS 端的应答消息时，会导致变迁 CallResp 点火执行，其附属的代码段也将同时执行。该代码段中的 st 与 sta 都是 SCENEC 类型的变量，st 为消息处理发生前的 UAC 状态，sta 为消息处理后的 UAC 状态。Action 部分直接调用 call_resp(st,resp)函数完成 UAC 端对不同类型应答消息的具体处理过程，该函数具体描述如下：

```
fun call_resp(st:SCENEC,resp:RESPONSES) =
  let
    val a=SCENEC.set_1 st RMFD;
    val b=SCENEC.set_3 a resp;
  in
    case resp of
      r2xx=>//消息类型为 r2xx 时的处理方式
        let
          val c=SCENEC.set_2 b NOREQU;
          val d=SCENEC.set_4 c (#5 st);
        in
          SCENEC.set_5 d TERM
        end
      r3xx=>//消息类型为 r3xx 时的处理方式
        let
          val c=SCENEC.set_2 b ACK;
          val d=SCENEC.set_4 c (#5 st);
        in
          SCENEC.set_5 d COMP;
        end
      _=>//消息类型为 r1xx 时的处理方式
        let
          val c=SCENEC.set_2 b NOREQU;
          val d=SCENEC.set_4 c (#5 st);
        in
          SCENEC.set_5 d PROC;
        end
  end
```

由上可知，UAC 端处理完类型为 r2xx 的应答消息后处于 TREM 状态，处理完类型为 r3xx 的应答消息后处于 COMP 状态，处理完类型为 r1xx 的应答消息后处于 PROC 状态。

综上，本节给出了 SIP 协议中 INVITE 事务的 TCPN 层次模型，对该协议关键数据、总体执行流程和网络环境、UAC

与 UAS 端具体功能等在不同的模型层次上分别描述。该模型具有数据结构完整、功能划分清晰直观、模型规模可控等特点，不仅为后续的协议验证与分析提供了良好的模型基础，还为基于层次 TCPN 对同类协议建模提供了实例参考。

4 基于 TCPN 模型的 SIP 协议验证与分析

基于第 3 节建立的 SIP 协议层次 TCPN 模型，本节首先对该模型能够满足协议中的时间约束条件进行了验证，之后综合利用模型模拟、状态空间分析、路径分析等模型分析技术[13]，确认协议模型是否满足协议功能规范要求，指出协议设计中存在的死锁缺陷，并对产生死锁的原因进行分析。

4.1 针对时间约束条件的验证

由于本文采用 TCPN 对 SIP 协议进行建模，定时规律作为时间约束关系的一个重要体现，需要验证其准确性，以保证最终模型能够正确模拟 SIP 协议行为。下面以模型 UAC 端的定时器 TimerA 和 TimerB 为例说明定时规律的验证过程，主要采用模型模拟执行的方式进行。依据 SIP 协议规范说明，当 TimerA 触发 6 次后，再经过 $T1$ 时间(表示往返时延)仍未收到应答，则会触发 TimerB，即 UAC 在 Calling 状态下，TimerA 至多触发 6 次，而 TimerB 至多触发 1 次。

首先，设置定时器触发的精确时间值作为参照比对，观察模型的模拟执行结果，若真实执行结果与参照比对值相同，则说明关于该定时器定时规律的建模是正确的。对定时器 TimerA 和 TimerB 触发时机的精确时间设置包括两方面：初值设置中 $T1=5$，协议执行初始时间为 0；设置 0 时刻开始发送 INVITE 请求，则 TimerA 第 1 次触发时间为 5，第 2 次触发时间为 15(2×5+5)，第 3 次触发时间为 35(4×5+15)，第 4 次触发时间为 75(8×5+35)，第 5 次触发时间为 155(16×5+75)，第 6 次触发时间为 315(32×5+155)，而 TimerB 的触发时间为 320(315+5)。这样，得到一组触发时间序列(5，15，35，75，155，315，320)，将该序列作为参照比对值，与模型实际的模拟执行结果进行比对。

前文构建的协议 TCPN 模型中，定时器的触发次数是随机的，例如 TimerA 的触发次数为[0，6]之间的随机整数。这样，假设在一次模拟中，TimerA 触发了 $m(0<m<6)$ 次，我们可以提取出 m 次的时间值与参照值做比对，但是对于其余 $6-m$ 次的触发规律却无从验证。因此，为了全面快速地对定时器进行验证，需要排除与定时器验证无关的因素的干扰。以 TimerA 和 TimerB 为例，先将图 3 中变迁 TransErr 防卫表达式中的 success()函数修改为：val successrate=0.0；fun success()=uniform(0.0，1.0) < successrate；这样，success()函数始终为假，描述传输层错误的变迁 TransErr 始终无法发生，避免了在验证过程中由于传输层错误的发生引发协议状态转移并进而导致无法验证只在 Calling 状态才有效的定时器 A 和 B 的定时行为。同理，将图 4 中变迁 CTOS 防卫表达式中的 success()函数修改为：val successraten=0.0；fun success()=uniform(0.0，1.0)<successraten；这样，success()函数始终为假，从而 CTOS 始终无法触发，相当于模拟了一条可靠率为 0 的链路，因此会不断地激活定时器超时事件，使我们能够有效完成对定时器触发行为中时间因素的实际观测。

基于上述设定，通过对模型进行模拟产生运行结果，图 4 中的库所 CollectorCTS 上带时间戳的 token 值清晰显示了每个消息的触发时间分别为 INVITE@0、INVITE@5、INVITE@15、INVITE@35、INVITE@75、INVITE@155、INVITE@315。其中 INVITE@0 表示初始条件下 UAC 所发出的 INVITE 请求，其余的 INVITE 消息都是由 TimerA 触发所引起的。由于 TimerA 负责控制消息的重传，因此可以从重传的消息所携带的时间值提取触发时间；TimerB 的触发会导致 UAC 状态的转移，故可以从描述 UAC 协议状态的库所 scenec 处提取其触发时间。通过观察可以看出时间变化规律与上文提取的参照比对值相同，由此验证了定时器 A 和 B 建模的正确性。

4.2 协议设计正确性验证

通过上述对定时器时间约束条件的验证，保证了模型在每种状态条件下，都可以按照预期设定的时间约束条件完成 SIP 协议功能。通过建模过程中不断执行的模型动态模拟，可以确认协议的 TCPN 模型与协议具体功能间的一致性，即保证模型本身对协议行为描述的正确性。基于此，通过模型状态空间分析技术，从模型是否存在活锁、死变迁、死锁等 3 个重要方面对协议设计进行更全面的正确性验证。死变迁的存在标识模型中有从未被点火的变迁，即模型存在建模冗余或缺陷。对于 SIP 协议的 INVITE 事务而言，其应正常终止于 UAC 和 UAS 都处在 Terminated 的状态，否则就认为系统执行过程出现死锁。活锁是指环形的状态空间，一旦进入其中，则无法离开，从而导致系统无法有效运行。

表 3 给出了第 3 节所建 SIP 协议 TCPN 模型的状态空间分析报告。分析结果可知，完全状态空间与其对应的强连通图含有相同数目的节点和弧，说明在不可靠链路条件下 SIP 协议的 INVITE 事务并没有活锁；同时，没有死变迁实例和活变迁实例等异常情况。

表 3 SIP 协议 TCPN 模型状态空间分析报告

分析内容	分析结果
Nodes(State Space)	4821
Arcs(State Space)	7494
Status(State Space)	full
Nodes(SCC Graph)	4821
Arcs(SCC Graph)	7494
Dead Transition Instance	None
Live Transition Instance	None
Dead Marking	249[786，778，767，758，694，…]

由于状态空间中的死标识(Dead Marking)节点较多，无法逐个在状态空间中显示并分析其是否为死锁状态，因此本文借助 CPN Tools 工具提供的查询函数(Query Function)对死锁状态进行进一步检测分析。表 4 给出 10 个查询函数(A0－A9)，用来判定不同条件被满足时死标识节点的数量。作为查询函数的核心部分，PredNodes 函数能够从给定的 Markings 集合中筛选出满足条件的死标识节点。如 A1 所示，第一个参数设置为 ListDeadMarking，表示对所有的死标识节点进行检测。第 2 个参数是筛选条件表达式，即 UAC 或 UAS 端正常结束。第 3 个参数通常设置为 NoLimit。位于最外层的 size 表示结果为满足设定条件的死标识节点的数量。上述所有死标识节点都满足 UAC 或 UAS 端至少一方正常结束(A0=A1)。UAC 和 UAS 端都正常结束的数量为 218(A2)，A2 与 A1 相差的 31 个死标识节点正是描述了只有 UAC 或 UAS 端一方正常结束的情况。A3=0 说明不存在 UAC 端正常结束而 UAS 端未正常结束的情况，即上述 31 个

死标识节点全部处于 UAS 正常结束而 UAC 非正常结束的状态(A4＝31)。A5＝A4＝31 说明 UAC 端均异常结束于 proceeding 状态。通过进一步查询定位可知,SIP 协议存在 4 种死锁可能性(对应于 A6－A9,且 31＝15＋2＋7＋7),即服务器端发送类型为 r2xx、r100、r101 或 r3xx 的应答时可能导致 UAC 端陷入死锁。

表 4　判定不同条件被满足时死标识节点数量的查询函数

	查询函数	结果	查询条件说明
A0	length(ListDeadMarkings())	249	整个模型 deadmarking 的数量
A1	size(PredNodes(ListDeadMarkings(), fn n=>(#5(hd(TMS. ms(Mark. Client'Scene 1 n)))=TERM orelse #5(hd(TMS. ms(Mark. Server'Scenes 1 n)))=STERM),NoLimit))	249	UAC 或 UAS 端只要一方正常结束
A2	size(PredNodes(ListDeadMarkings(), fn n=>(#5(hd(TMS. ms(Mark. Client'Scene 1 n)))=TERM andalso #5(hd(TMS. ms(Mark. Server'Scenes 1 n)))=STERM),NoLimit))	218	UAC 和 UAS 端都正常结束
A3	size(PredNodes(ListDeadMarkings(), fn n=>(#5(hd(TMS. ms(Mark. Client'Scene 1 n)))=TERM andalso #5(hd(TMS. ms(Mark. Server'Scenes 1 n)))<>STERM),NoLimit))	0	UAC 端正常结束,且 UAS 端没有正常结束
A4	size(PredNodes(ListDeadMarkings(), fn n=>(#5(hd(TMS. ms(Mark. Client'Scene 1 n)))<>TERM andalso #5(hd(TMS. ms(Mark. Server'Scenes 1 n)))=STERM),NoLimit))	31	UAS 端正常结束,且 UAC 端没有正常结束
A5	size(PredNodes(ListDeadMarkings(), fn n=>(#5(hd(TMS. ms(Mark. Client'Scene 1 n)))=PROC andalso #5(hd(TMS. ms(Mark. Server'Scenes 1 n)))=STERM),NoLimit))	31	UAS 端正常结束,且 UAC 端以 PROC 状态结束
A6	size(PredNodes(ListDeadMarkings(), fn n=>(#5(hd(TMS. ms(Mark. Client'Scene 1 n)))=PROC andalso #5(hd(TMS. ms(Mark. Server'Scenes 1 n)))=STERM andalso #2(hd(TMS. ms(Mark. Server'Scenes 1 n)))=r2xx),NoLimit))	15	UAS 端正常结束,UAC 端收到服务器端类型为 r2xx 应答时导致的异常结束
A7	size(PredNodes(ListDeadMarkings(), fn n=>#5(hd(TMS. ms(Mark. Client'Scene 1 n)))=PROC andalso #5(hd(TMS. ms(Mark. Server'Scenes 1 n)))=STERM andalso #2(hd(TMS. ms(Mark. Server'Scenes 1 n)))=r100),NoLimit))	2	UAS 端正常结束,UAC 端收到服务器端类型为 r100 应答时导致的异常结束
A8	size(PredNodes(ListDeadMarkings(), fn n=>(#5(hd(TMS. ms(Mark. Client'Scene 1 n)))=PROC andalso #5(hd(TMS. ms(Mark. Server'Scenes 1 n)))=STERM andalso #2(hd(TMS. ms(Mark. Server'Scenes 1 n)))=r101),NoLimit))	7	UAS 端正常结束,UAC 端收到服务器端类型为 r101 应答时导致的异常结束
A9	size(PredNodes(ListDeadMarkings(), fn n=>(#5(hd(TMS. ms(Mark. Client'Scene 1 n)))=PROC andalso #5(hd(TMS. ms(Mark. Server'Scenes 1 n)))=STERM andalso #2(hd(TMS. ms(Mark. Server'Scenes 1 n)))=r3xx),NoLimit))	7	UAS 端正常结束,UAC 端收到服务器端类型为 r3xx 应答时导致的异常结束

4.3　协议死锁行为的原因分析

为了详细分析上述 4 种死锁导致的协议执行异常终止,需要首先在模型状态空间中定位这些异常 Marking,之后提取从初始 Marking 到异常 Marking 的执行路径,分析路径中的 Marking 数据和点火绑定,以此获知 UAC 和 UAS 进行了怎样的交互行为才导致了死锁的产生。

A6－A9 描述的每种死锁情况都含有多个死标识节点(A6 为 15 个、A7 为 2 个、A8 和 A9 为 7 个)。为了便于分析,需要明确每个死标识节点在状态空间中的具体数据信息,我们仍使用 CPN Tools 工具提供的查询函数来获得标识。以 A6 描述的情况为例,查询函数如下:

```
PredNodes(ListDeadMarkings(),
  fn n=>(#5(hd(TMS. ms(Mark. Client'Scene 1 n)))=PROC and
      also#5(hd(TMS. ms(Mark. Server'Scenes 1 n)))=
      STERM andalso#2(hd(TMS. ms(Mark. Server'Scenes 1
      n)))=r2xx),NoLimit)
```

函数的运行结果为[108,1135,1516,1521,1785,1786,2627,2716,3321,3327,571,61,62,758,767],表示导致 A6 所示死锁的各个状态空间节点,不失一般性,我们选择其中的 108 号节点进行后续的分析说明。

接下来的工作是从状态空间中提取从初始节点到 108 号节点的路径。首先,利用 CPN Tools 中提供的 NodeInPath 函数提取状态空间初始节点到 108 号异常节点的最短路径所经过的 Marking 值,并将其存到临时文件中。接着,将该路径中涉及 UAC 和 UAS 的部分分离,分别得到协议在 UAC 端和 UAS 端运行过程中产生的状态数据序列,这两个序列分别描述了 UAC 端和 UAS 端从初始状态到异常终止状态的执行过程和状态变化过程。最后,为了便于更为直观地分析死锁产生的原因,需要将上述序列间的交互细节以图形化的方式展现出来。本文首先采用正则表达式对上述两个序列分别进行过滤与格式处理,可生成如下所示的两个文件(实际上为有限状态机描述的动作序列),之后将这两个文件作为绘图工具 Graphviz[14] 的输入,可以生成图 6 所示的 UAC 与 UAS 的交互顺序图。

```
ss. dot(UAS 端对应的执行序列)
digraph finite_state_machine
{
  rankdir=LR;size="8,5";node [shape=circle];
  node [shape=doublecircle];STERM;
  SINI->SINI[label="1-NOMESS-NORESP-NOREQU"];
  SINI->SPROC[label="2-RMFD-NORESP-INVITE"];
  SPROC->SPROC[label="3-T200-r100-INVITE"];
  SPROC->SPROC[label="4-RMFD-r100-INVITE"];
  SPROC->STERM[label="5-RMFU-r2xx-INVITE"];
}
```

```
cs. dot(UAC 端对应的执行序列)
digraph finite_state_machine
{
  rankdir=LR;size="8,5";node [shape=circle];
  node [shape=doublecircle];TERM;
  INI->CALL[label="1-RMFU-INVITE-NORESP"];
  CALL->PROC[label="2-RMFD-NOREQU-r100"];
  PROC->PROC[label="3-RMFD-NOREQU-r100"];
}
```

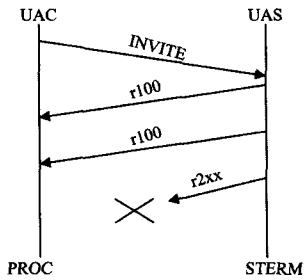

图 6　导致死锁产生的 UAC 与 UAS 交互顺序图

分析图 6 可知，因 UAS 的协议上层模块在一定时间段内没有产生应答而成功重发 r100 后，UAC 仍处于 PROC 状态，但当 UAS 端在发送 r2xx 的应答消息后，其终止本次事务执行并转入终止状态，但是该消息由于不可靠链路的原因而丢失，UAC 并未收到，且 UAC 仍继续处在 PROC 状态无法正常终止，从而造成死锁。文献[9,10]也指出了这种死锁。对其他 3 种死锁进行同样的分析可发现：UAS 端发送的应答数据包丢失、UAS 端在 PROC 状态时发生传输层错误是导致死锁产生的原因，因为这时 UAS 端均结束本次事务执行而转入终止态，但 UAC 仍在 PROC 状态等待而无法正常终止。

5　SIP 协议设计的改进方案

SIP 协议存在的死锁状态，将导致协议无法完成正常功能，且 UAC 持续等待 UAS 应答时，攻击者可以通过事先截获通信内容，冒充 UAS 与 UAC 进行后续交互，对 SIP 协议的安全也构成了威胁。本节首先根据在死锁状态下 UAC 与 UAS 不同的行为特点提出 3 种 SIP 协议设计改进方案，然后基于 TCPN 对方案的正确性和可行性进行验证，最后通过方案对比明确各自的适用场合。

5.1　改进方案

解决死锁问题可以从改进 UAC 端的协议设计、改进 UAS 端的协议设计、同时改进 UAC 和 UAS 端的协议设计等 3 个角度来考虑，下文给出各自的解决方案并进行验证。

· 方案 1：改进 UAC 端的协议设计

使 UAC 端的协议设计具备接收协议上层模块指令的能力，即 UAC 上层模块发送“BYE FROM TU”的终止死锁消息，这样借助上层模块的判断结束 UAC 在 proceeding 状态的等待，消除死锁。上层模块可以根据会话参与者的数量、传输数据的类型、服务质量的要求，对判断死锁的时机做出灵活动态的调整，这样一来不仅可以解决死锁问题，还可以适应不同通信环境下对协议的要求。

· 方案 2：改进 UAS 端的协议设计

依据原有的协议设计规范，UAS 端发送应答后将直接从 proceeding 状态转入 terminated 状态，而一旦进入该终止态，UAS 端将不会回送任何消息。所以，改进协议设计使 UAS 发送应答后继续留在原状态，并报告 UAS 协议上层模块，这样延长了 UAS 在 proceeding 状态下的等待时间和整个会话时间，使得原会话得以继续，从而降低了出现死锁的概率。

· 方案 3：同时改进 UAC 和 UAS 端的协议设计

方案 3 是方案 1 与方案 2 的组合，将 UAC 端采用的上层模块通知与 UAS 端采用的会话延时结合起来，这样可以更有效地避免死锁产生，增强协议可靠性和可用性。

5.2　改进方案的正确性验证

为了验证上述各方案是否能够有效避免死锁产生，需要依据每种方案的详细设计对 SIP 协议的原 TCPN 模型进行相应修改，并再次执行 4.2 节所述的验证方法。

· 改进方案 1 的正确性验证

在图 2 所示 TCPN 模型的 UAC 部分添加 ChandleDeadlock 模块来模拟 SIP 协议上层模块的行为，如图 7 所示。为简单起见，对死锁的处理主要通过 ProceedingDeadlock 变迁点火，其附属代码段 proc_trr(st)同时执行，该函数定义为：

```
fun proc_trr(st:SCENEC)=
  let
    val a=SCENEC. set_1 st TU_BYE;
    val d=SCENEC. set_4 a (#5 st);
  in
    SCENEC. set_5 d TERM
  end
```

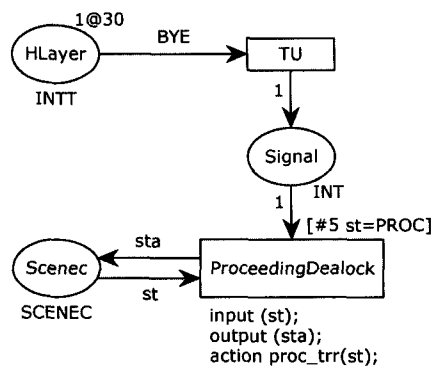

图 7　改进方案 1 中 SIP 协议上层模块行为页

模型中库所 HLayer 控制 BYE 消息发送的时机，也就是上层模块对于死锁的反应时间，可以通过模型模拟找出发生时间最晚的状态点，以该时间作为反应时间的估计值(本文经试验取值 66)。这样做是因为一旦发送 BYE 消息，则会终止本次会话，如果选择较早的死锁时间，则 BYE 消息的提早发送会导致状态空间生成不充分，进而影响后续的分析结果。

执行 4.2 节所述的验证方法可知，完全状态空间与对应的强连通图含有相同数目的节点(6086)和弧(10531)，说明没有活锁，且没有死变迁和活变迁实例等异常情况。模型共产生 210 个死标识节点，且其全部属于 UAC 端与 UAS 端正常终止的情况，由此证明采用上层模块发送 BYE 消息的方案可以防止 UAC 在 PROC 状态长时间等待，避免了死锁的发生。

· 改进方案 2 的正确性验证

为了模拟发生传输错误时 UAS 端状态的延迟变化，需要将原模型中控制状态变化的 stran_err 函数更改如下：

```
fun stran_err(sst:SCENES) =
```

```
if #5 sst=SPROC then //保持状态不变,为 PROC 状态
  let
    val a =(#4 sst);
    val b=SCENES. set_1 sst TRR;
    val c=SCENES. set_4 b a;
  in
    SCENES. set_5 c SPROC
  end
else //原模型中改变状态的部分
  let
    val a=(#4 sst);
    val b=SCENES. set_1 sst TRR;
    val c=SCENES. set_4 b a;
  in
    SCENES. set_5 c STERM
  end
```

执行 4.2 节所述的验证方法可知,完全状态空间与对应的强连通图含有相同数目的节点(12340)和弧(21645),说明没有活锁,且没有死变迁和活变迁实例等异常情况。模型共产生 437 个死标识节点,且其全部属于 UAC 端与 UAS 端正常终止的情况,由此证明在 UAS 端采用延迟状态变化的方案可以避免死锁的发生。

· 改进方案 3 的正确性验证

协议 TCPN 模型采用层次化建模,使用函数来描述状态变化,便于独立更改各个模块功能,因此,可以直接合并两种解决方案对模型的上述修改。通过再次验证可知,完全状态空间与对应的强连通图含有相同数目的节点(6297)和弧(10893),说明没有活锁,且没有死变迁和活变迁实例等异常情况。模型共产生 216 个死标识节点,且其全部属于 UAC 端与 UAS 端正常终止的情况,由此证明同时使用方案 1 和 2 也可以有效避免死锁发生。

5.3 改进方案的比较

文献[9,10]在 UAC 端的 proceeding 状态下添加定时器,一旦定时器超时,则终止本次会话,以此避免死锁发生。因此,设置科学合理的定时器超时时间成为核心问题,定时器时间设置得太长,则会延长本次会话的终止时间,造成 UAC 端的无谓等待;如果设置得太短,则会在正常会话未完成的情况下强行终止会话,使高层必须重新开启一个会话来完成后续交互,加重通信双方的资源消耗。但是文献[9,10]并未给出定时器时间值的详细说明。与之相比,本文提出的采用高层通知、适当延长 UAS 等待时间等方式来避免死锁产生的方法避免了上述弊端,可以在有效解决死锁造成的协议可用性问题的同时不再引入新的问题。

本文提出的 3 种方案在改动原协议程度、部署难度等方面也各有优劣。其中,方案 1 采用 UAC 从上层模块被动接收消息的方式来避免死锁,并未涉及协议状态的转移以及消息的发送,因此对原协议的改动程度最小,但对于支持大量移动设备的应用来说部署难度较大。方案 2 采用 UAS 端延长状态变化时间的方式避免死锁,对原协议的改动程度较大,但仅对服务器端的协议进行改进,故部署较为容易。方案 3 是 1 和 2 的联合使用,故对原协议的改动程度最大,部署难度也最高,但能够更全面地保障 SIP 协议稳定可靠地运行。综上,在完善现有 SIP 协议应用或研发新的 SIP 协议应用时可以依据实际应用条件选择不同的方案加以实施,以有效增强 SIP 协议在实际应用中的可行性和可靠性。

结束语 本文充分利用 TCPN 模型在描述和分析具有复杂交互行为及时间约束的系统方面的优势,为在移动通信网络中广泛使用的 SIP 协议构建了层次 TCPN 模型,正确描述了 UAC 向 UAS 直接呼叫的协议核心功能行为。集成模型模拟、状态空间分析、路径分析等多种模型分析技术,对时间约束条件和协议正确性进行了有效验证,并指出了能够导致死锁产生的 SIP 协议设计缺陷。在分析产生死锁原因的基础上,提出了 3 种可用的协议设计改进方案,并验证了改进方案的正确性,在不同适用场景下选择采用合适的改进设计方案,能够有效增强 SIP 协议在实际应用中的可行性和可靠性。

参 考 文 献

[1] Rosenberg J, Schulzrinne H, Camarillo G, et al. RFC 3261, SIP: Session Initiation Protocol [S]. Internet Engineering Task Force, 2002

[2] 张智江,张云勇,刘韵洁. SIP 协议及其应用[M]. 北京:电子工业出版社,2005

[3] Jensen K, Kristensen L M. Coloured Petri Nets: Modelling and Validation of Concurrent Systems[M]. Berlin: Springer, 2009

[4] 王东敏. 基于 Petri 网的 SIP 协议一致性测试套的设计与实现[D]. 北京:北京邮电大学,2011

[5] 郝建国,邹嘉,戴一奇. 基于 Hash 链的 SIP 服务实时支付方案[J]. 清华大学学报:自然科学版,2009,49(12):581-585

[6] 陈效庭. 基于 SIP 协议的 IP 智能网的应用及容灾测试分析[D]. 上海:上海交通大学,2010

[7] Paolo D, Jaume N, Josep M, et al. Interworking Scheme Using Optimized SIP Mobility for Multi-Homed Mobile Nodes in Wireless Heterogeneous Networks[C]// Proc. of the IEEE 71st Vehicular Technology Conference. Taipei, China, 2010: 1-6

[8] Boucadair M. Migrating SIP-based Conversational Services to IPv6: Complications and Interworking with IPv4[C]// Proc. of the 2nd International Conference on Digital Telecommunications. San Jose, USA, 2007: 2-3

[9] Ding L G, Liu L. Modelling and analysis of the INVITE transaction of the session initiation protocol using coloured Petri nets [C]// Proc. of 29th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency. Xi'an, China, 2008: 132-151

[10] Liu L. Verification of the SIP transaction using coloured Petri net[C]// Proc. of the 32nd Australasian Computer Science Conference. Wellington, New Zealand, 2009: 75-84

[11] Gehlot V. Colored Petri Net model of the Session Initiation Protocol(SIP)[C]// Proc. of 36th Annual Conference on IEEE Industrial Electronics Society. Phoenix, USA, 2010: 2150-2155

[12] 杨鹏,袁占亭,王继曾. 基于广义随机 Petri 网的 SIP 的验证和性能分析[J]. 系统仿真学报,2007,19(S1):151-154

[13] Westergaard M. CPN Tools[OL]. http:// cpntools. org, 2013

[14] Graphviz[OL]. http://www. graphviz. org, 2013