**Report on SynbexNet**

SYMBEXNET: Testing Network Protocol Implementations with

Symbolic Execution and Rule-Based Specifications

本文主要设计了一个面向于网络协议的软件系统分析系统SynbexNet。SynbexNet是基于符号执行（symbolic execution）和规则检测（rule-based specification）的工具。

**本文的主要创新点如下：**

（1）在符号执行和规则检测的基础上设计了S2E平台。

1. 分别针对状态空间不同的网络协议设计了单/多数据包交换符号执行算法（Single/Multi-Packet Exchange Symbolic Execution）。

2. 引入了规则提取和规则检测步骤。

（2）完成了平台的实现，并对其进行运行测试。

1. 测试了Zeroconf和DHCP协议的不同实现形式中存在bug。

**本文的主要解决思路如下：**

**1. SynbexNet面向的问题：网络协议**

网络协议一般由协议（及其说明）和协议实现两部分组成。协议描述了网络协议的规则，然后在不同的应用环境中，会根据不同的需求产生不同的网络协议实现程序。对于同一个协议，不同的协议实现程序由于开发者不同，应用环境不同，往往会产生一些不同的错误。而要追踪并处理这个错误面临着很多挑战。

（1）网络协议实现的一些错误，往往只能在长时间处理数据包之后才能被检测到。

（2）由于协议实现是针对不同的应用环境，基于相应的网络协议而设计的。因此保证协议实现实现的规则和原始的网络协议相匹配是协议实现软件分析的一个重要的目标。而要自动化的验证规则是否满足，首先需要能够将规则提取并表示出来。此前的工作缺乏这一部分的研究。

（3）对于复杂的网络协议实现而言，其状态空间往往很大，因此计算复杂度很大，容易发生指数爆炸。

**2. SynbexNet如何解决这些问题？**

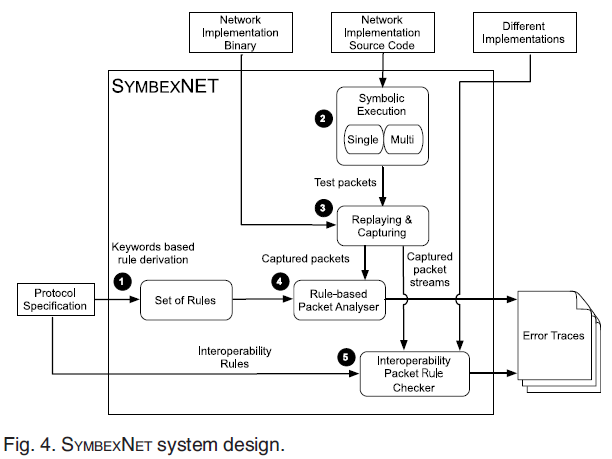
（1）针对网络协议实现的问题，SynbexNet会连续的向协议实现发出数据包，藉此来获取不同的错误类型。

（2）针对网络协议实现状态空间的问题，SynbexNet分别针对于无状态的（stateless）和多状态（stateful）的网络协议设计了单/多数据包交换符号执行算法（Single/Multi-Packet Exchange Symbolic Execution）。

（3）对于网络协议的规则验证的问题，文中给出规则提取，规则表示和基于符号验证的规则验证方法。

（4）给出了网络协议的不同协议实现实现之间的互操作性测试。

一个完整的SynbexNet的流程图可以表示为下图。其中，主要有两路输入，一路自上而下，是符号执行的部分。SynbexNet根据网络协议的实现代码（高级语言）使用KLEE进行符号执行，然后生成一组测试输入（step2）。另一组是自左向右的，根据协议说明文档，提取出协议的规则（step1）。然后根据提取出来的规则和协议的内容生成一个基于规则的数据包分析模块（step4），和一个互操作性验证模块（step5）。得到了这些基本单元以后，SynbexNet将生成的测试输入输入到网络的视线中进行重新运行（replay）并提取出其中的信息（step3）。再将这部分信息用step4和step5中得到的分析模块进行分析。最终给出出错的路径。



**3. SynbexNet的执行细节：**

SynbexNet的结果主要是两个部分，符号执行部分和规则分析部分。其中前者主要是给出了两个针对于不同协议类型的符号执行算法。后者则是一套专门针对于网络协议软件规则的分析方法。

**（1）单数据包交换符号执行（SPE-SE）**

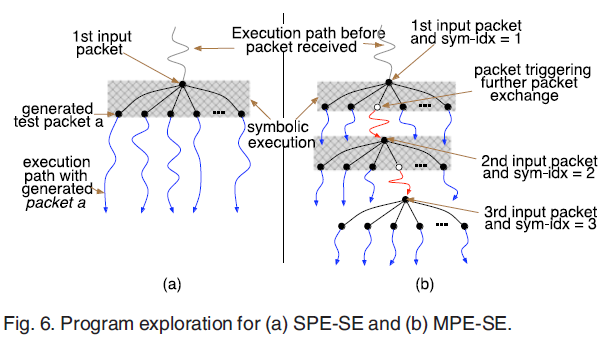
如标题的含义，SPE-SE就是使用一个单独的符号化的输入数据包来进行符号执行的方法。这种方法是面向无状态的网络协议而设计的。这种网络协议处理数据包是相对独立的。

**Step1. 测试数据包生成**

这里是将测试数据包视为变量。当符号化的数据包输入到网络协议实现中时，SynbexNet就将数据包中知道的区段用符号表示，然后运行KLEE来完成符号执行。最终得到相应的测试数据包集。

**Step2. 测试数据包重运行（replay）**

简单的说，就是将测试数据包用client重新输入到协议实现里面去，然后SynbexNet会记录client和协议实现的全部输入输出。这些信息最终被用来做规则验证的输入。



**（2）多数据包交换符号执行（MPE\_SE）**

MPE-SE主要是面向于多状态的网络协议而设计的。后者的主要特点是存在一些情况，如错误，只能在多个测试数据包连续输入的时候才会产生。因此SPE-SE不能获得这种协议的全部运行情况。（文中给出了一个范例，中间有一个判断分支，需要前一个输入数据包）

解决这个问题的办法就是用多输入数据包的符号化来解决。但是这种方式需要遍历的状态空间太大了（指数的）。因此，文中的MPE-SE给出符号+具体数据混合表示的方法来降低复杂度。MPS-SE是一个递进的算法，其中第k轮执行的主要流程如下：

Step1. 具体数据执行

使用之前k-1轮给出数据作为具体数据

Step2. 符号执行

第k个输入数据包使用符号化数据包作为输入。

Step3. 按照KLEE的流程，执行并终止。

**（3）规则检查**

规则检查是网络协议程序中特有的部分。由于前文中的符号执行可以完成对底层代码的逻辑或语法问题进行检查，规则检查补充了对网络协议实现和协议本身之间的规则匹配性的检查。（语义错误）

Part1. 规则提取

给出了什么样的规则可以被提取。一般讲，协议视为黑盒子，不涉及到内部状态的规则都可以被提取出来。这是因为规则可以被不同的协议实现共享，所以不能涉及到实现的内部状态。另一方面，这样的做法也降低了规则提取的复杂度。

Part2. 基于规则的数据流语言

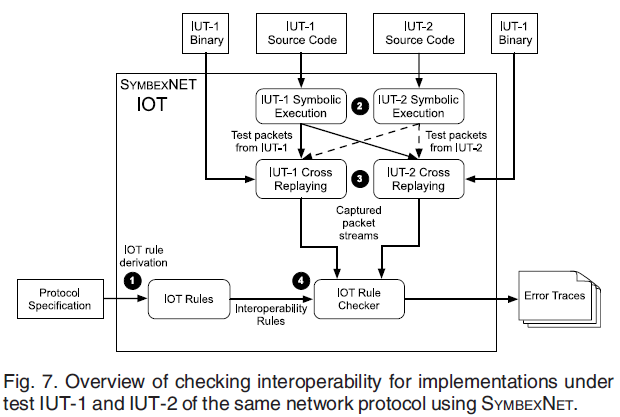
设计了数据包模式匹配语言。设计了与之匹配的表达形式，规则运算符，变量组合符号，时间参数。

Part3. 规则建立和验证

使用一个非确定性有穷自动机——这一小段看不懂了。

**（4）互操作性检测（IOT）**

这一部分是针对不同的协议实现之间的互操作性来设计的。如图所示：



互操作性检测就是基于符号执行和规则检测的。其中两个不同的协议实现交换其产生的测试输入数据包，然后将提取出来的数据流作为互操作性检测的输入。而在互操作性规则检测这一部分，首先需要提取互操作规则，然后根据该规则建立规则检测模块。最终的输出结果是出现互操作性的错误路径。

如前文所述，可选择的符号执行就是对于环境条件中产生的分支路径进行剪枝。这里的剪枝并不是删掉应用的分支，而是将环境条件中的分支打包起来。具体的打包方法就是使用一组真实数据来运行这段代码。而这组真实的数据实在环境条件被调用时的路径约束所约束的。S2E处理真实数据的数据执行模式和符号执行模式之间的切换主要分为四个过程。

例如，在图中例子里，app代码是环境条件，需要进入数据执行的模式。其之前没有路径约束（app层是路径的起点），那么对于一个变量x，可以选择一个合理的x=5来执行app层。

**阶段一：环境条件调用处理单元**

当app层调用lib层时，lib层是处理单元，因此需要将变量符号表示，进入符号执行的模式。此时，x需要从真实数据5变为符号a。其中a的约束条件由当前保存的路径约束条件得到——例子中是运行的开始，因此条件应为全体实数。由于x在lib中完成了符号执行。因此，路径在lib的代码段中产生了分叉。

需要说明的是，与此同时根据x=5的真实数据也会并行的写入lib层代码中，进行真实数据执行。下面的阶段中，这一个真实数据的路径会始终保存，且唯一。

**阶段二：处理单元调用环境条件**

当lib代码中开始调用kernel层时，变量x再一次进入了环境条件中。此时需要根据x所处于的路径约束构建一个x的值。因此，在环境条件kernel层中，用于测试的是数据执行模式。

**阶段三：环境条件返回处理单元**

Kernel层完成数据处理，将数据返回lib层。代码从环境条件返回到处理单元中，测试模式需要从数据执行模式跳转回符号执行模式。此时需要根据x的值构建一个x的约束条件。再将新的变量约束条件加入lib层的符号执行中。

**阶段四：处理单元返回环境条件**

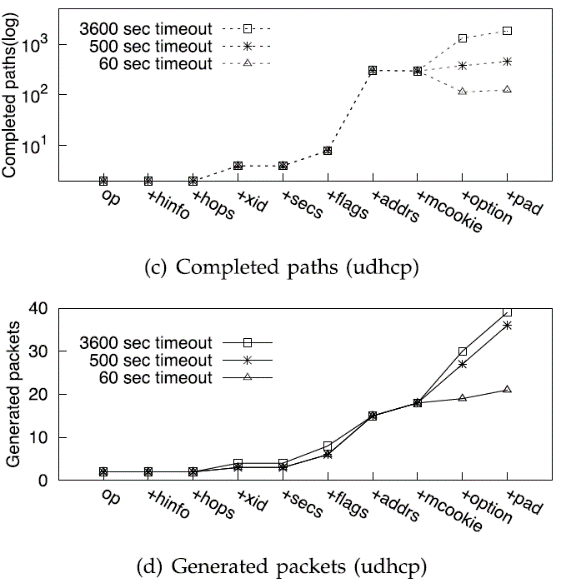
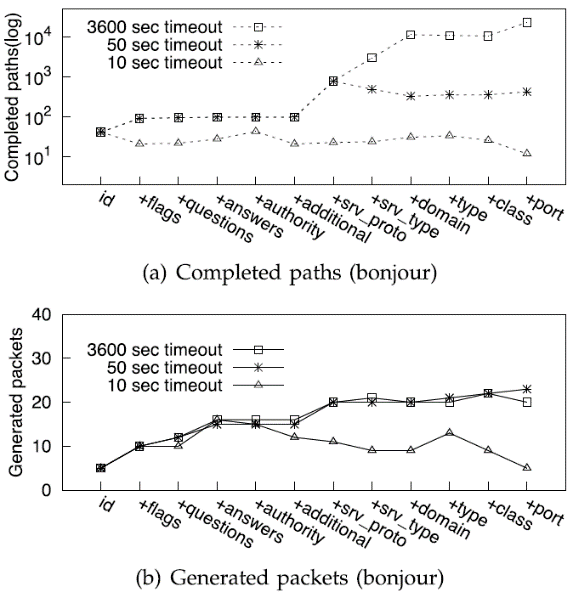
处理单元处理好数据，将数据返回给环境条件。此时的路径约束条件会保存在环境条件处。而环境条件中所使用的真实数据来源于最早的x=5数据执行到此处的结果。实际上，这组结果一定是满足对应的约束条件的。这组约束条件会在下一个阶段一出现时用于构建变量所需的约束条件。

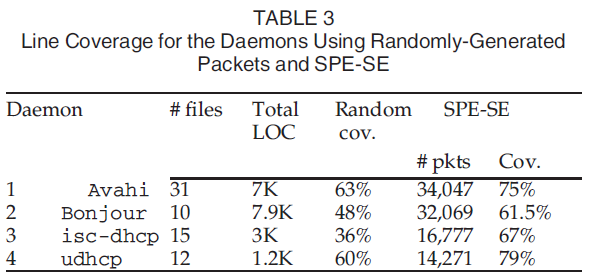
**本文的实验验证如下：**

实验主要是在五个不同的协议（3个Zeroconf，2个DHCP）实现上运行SynbexNet。然后测试其覆盖率，验证其规则和互操作性。

**1. 验证SCE-SE**

首先确定需要符号化多少个域（field）。如下图。存在一个性能和精确度的折中。

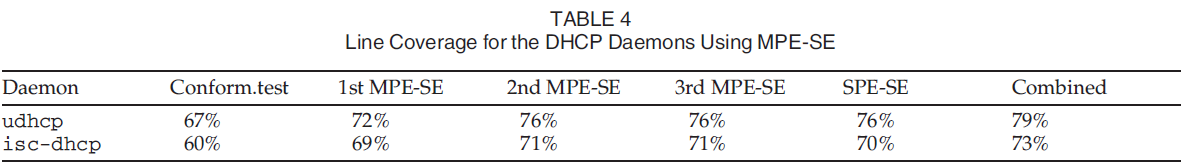




上图给出了SynbexNet基于D-ITG方案的提升。

**2. 验证MCE-SE**

SCE-SE和MCE-SE的对比实验，感觉两个方案没有什么区别。



问题一：SPE-SE和1stMPE-SE居然不一样？

问题二：MPE-SE不应该是可以覆盖SPE-SE的吗？那两者的组合是怎么回事？

**3. 互操作性测试**

给出SynbexNet的测试结果，但是没有对比试验。

**4. 错误发掘**

给出了一些发现的错误——但是没有对比。