Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

协议模糊测试方法

硕士研究生 徐菊彬

2026年02月08日

问题回溯

- 问题回溯
  - 前期讲授不流畅，后期语速过快
  - 对算法讲解不深入，浮于表面
- 相关内容
  - 2024.09.03 张浩然《大模型赋能的模糊测试用例生成技术》
  - 2024.12.23 徐菊彬《网络未知协议逆向技术》
  - 2025.05.18 徐菊彬 《大模型指导的协议模糊测试》

内容提要

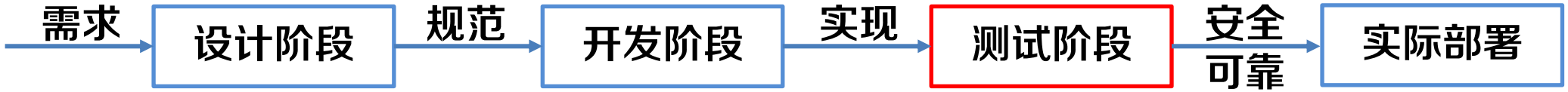
- 预期收获
- 题目内涵解析
- 研究背景与意义
- 研究历史与现状
- 知识基础
- 算法原理
  - HSPFuzzer
  - Fuzztruction-Net
- 特点总结与工作展望
- 参考文献

预期收获

- 预期收获
  - 1. 了解协议模糊测试方法的基本概念和研究方向
  - 2. 理解两种协议模糊测试方法的基本原理
  - 3. 了解现有方法的缺陷以及未来发展方向

内涵解析与研究目标

- 内涵解析
  - 网络协议：计算机网络中进行数据交换的规则，保证节点之间的相互通信



  - 模糊测试：生成大量随机突变测试用例，旨在触发软件程序中异常运行时的行为
  - 协议模糊测试：通信复杂性高、测试环境相对受限
- 研究目标
  - 对协议实现的服务端或客户端源代码进行模糊测试
  - 提高协议模糊测试的效率和有效性
    - 提高模糊测试的吞吐量（单位时间内执行模糊测试的次数）
    - 改进种子调度和变异策略，扩大对协议状态、源代码的覆盖率

数据生成
数据传输
迭代效率
速度
质量

研究背景与意义

- 研究背景
  - 网络协议是互联网通信的基础，协议实现的复杂性使得开发过程中易引入漏洞，导致严重的安全问题
    - Heartbleed漏洞存在于OpenSSL的TLS协议实现中，在实现TLS的心跳扩展时没有对输入进行适当验证（缺少边界检查），导致缓冲区过读，影响了全球超过17%的服务器
  - 传统模糊测试在协议场景中具有局限性，需针对协议规范设置结构化测试用例，实现状态空间的有效探索
- 研究意义
  - 研究高效的协议模糊测试技术，在软件发布前或部署早期主动发现漏洞，提升网络基础设施安全性与健壮性

研究历史与现状

Pham等人提出AFLNET，第一个用于协议实现的灰盒模糊器，将消息序列视为覆盖定向灰盒模糊测试的种子，将每条消息与相应的协议状态相关联，并最大化状态空间和代码的覆盖范围，是首个使用代码和状态覆盖引导的协议模糊器

2020

2023

Qin等人提出基于程序变量的状态表示方案和有效的交互同步机制来提高模糊测试效率NSFuzz，通过手动注释消息处理的结束来减少延迟，使用静态分析和注释应用程序编程接口（API）来识别服务内的同步点和状态变量，实现快速 I/O 同步和准确的服务状态跟踪

Meng等人提出大模型指导的模糊测试方法CHATAFL，将大语言模型与经典协议模糊测试器AFLNet相结合，利用LLMs学习协议规范文档RFC，指导生成合规且多样化的输入数据，同时设置覆盖高原检测机制，提高了覆盖率和漏洞发现效率

2024

2024

Fu等人提出HNPFuzz，客户端和服务器之间的测试用例和响应消息通过共享内存传输，由精确的同步器引导，而不是通过套接字接口，缩短迭代周期。并设计了一种持久模式的模糊测试，消除了每个输入的 SUT 重新启动

Bars等人提出了一个基于故障注入的协议模糊方法Fuzztruction-Net通过修改SUT的对等点（例如，模糊测试服务器时的客户端）来引入故障注入，以确保变异消息通过完整性和加密检查

2024

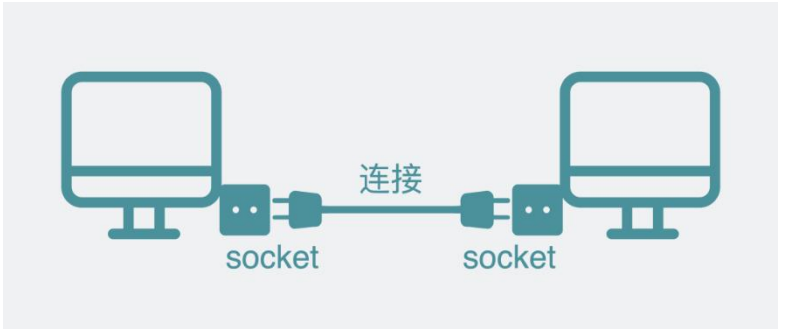
2025

Xia等人提出HSPFuzzer，一种高速协议模糊器，它利用连接重用来减少 SUT 重新启动和连接重新建立。采用前缀消息识别算法来确定连接内所需的基本数据包，并采用了覆盖监控机制来检测异常执行状态

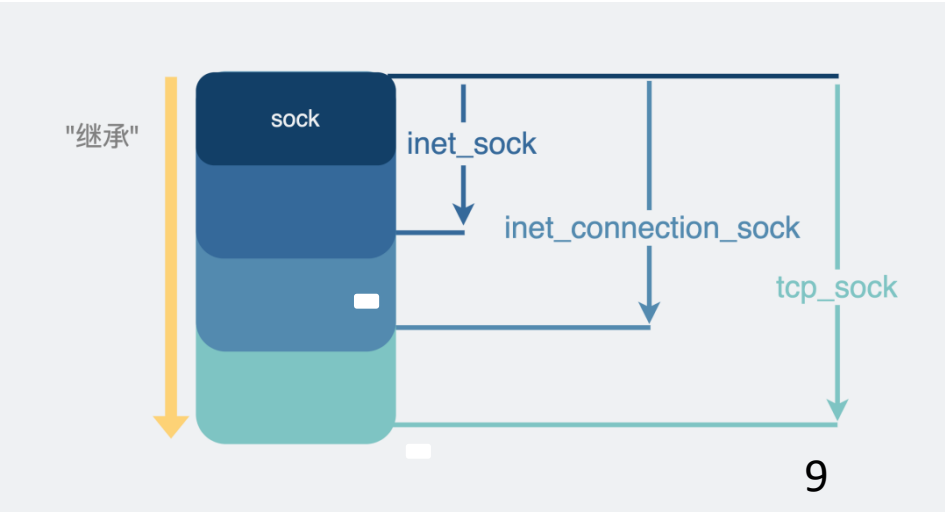
- socket套接字
  - 网络编程中的一种通信机制，支持TCP/IP的网络通信的基本操作单元
- 通信过程实例

知识基础

- socket套接字
  - 代码库 or 接口层，介于内核和应用程序之间
  - 提供了一些高度封装过的接口，使用内核网络传输功能



- sock，位于操作系统内核中
  - Sock：最基础的结构，维护收发数据缓冲区
  - inet_sock：网络传输功能的sock，添加端口，IP等字段信息
  - inet_connection_sock：面向连接的sock
    - 在inet_sock的基础上加入面向连接的协议里相关字段，如accept队列，数据包分片大小，握手失败重试次数等
  - tcp_sock：tcp协议专用的sock结构
  - udp_sock：udp协议专用的sock结构

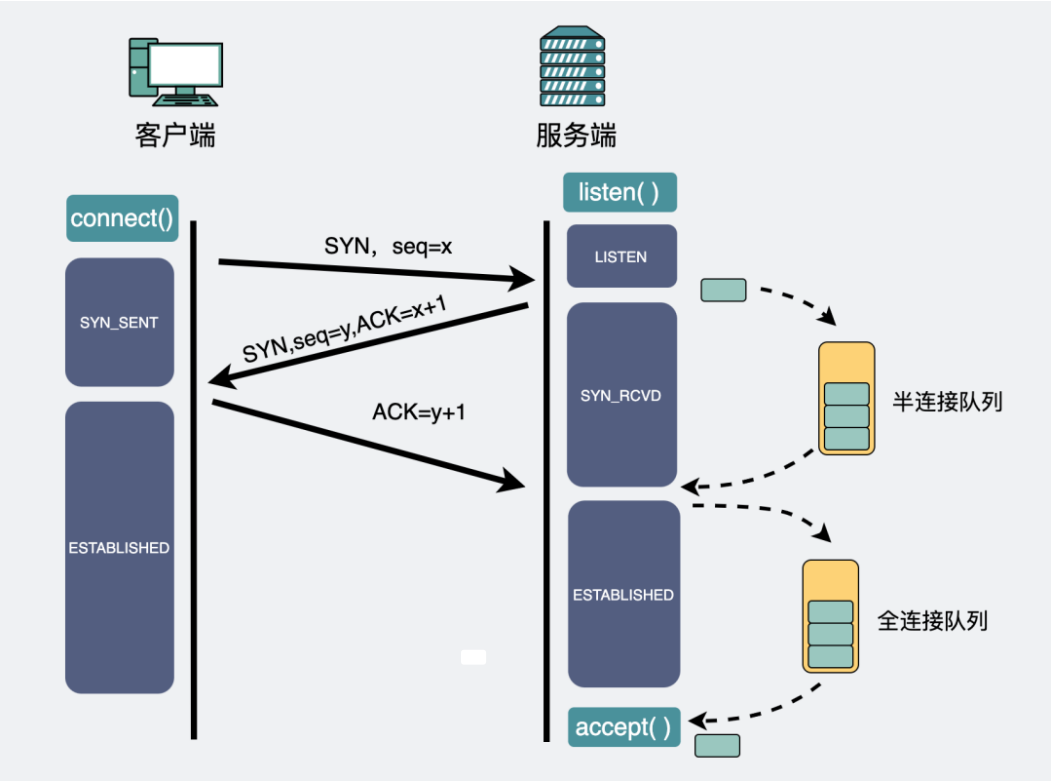


- socket套接字
  - 在用户空间使用文件句柄socket_fd，来操作内核sock的网络传输功能

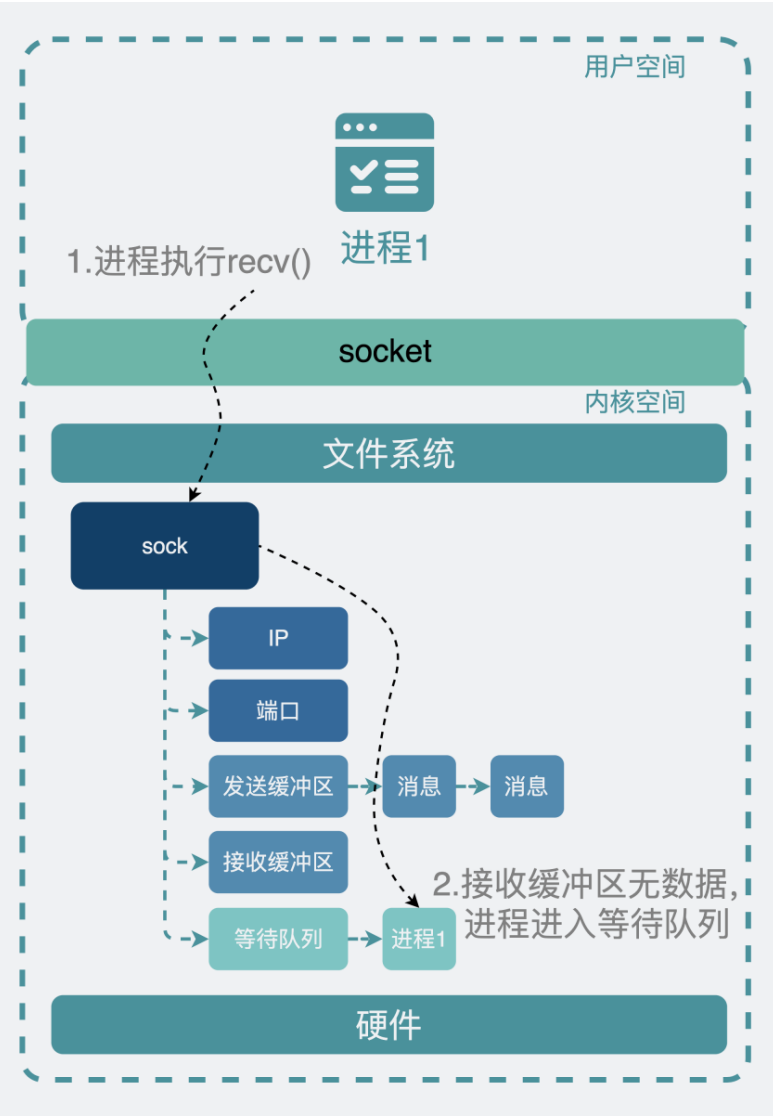
```
socket_fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)
```

socket如何实现网络通信—连接建立&数据传输

- 连接建立-TCP三次握手
  - 客户端中，执行$connect(socketfd, "ip:port")$
    - 根据句柄找到文件进而指向内核sock结构
    - 主动发起三次握手
  - 服务端中，执行$listen()$时创建两个队列
    - 半连接队列（握手数<3）
    - 全连接队列（完成三次握手的连接）



- 数据传输
  - 执行$send()$发送数据
    - 根据句柄找到对应文件，找到文件指向的sock结构
    - 将数据放到sock结构中的发送缓冲区，结束流程
    - 内核“看心情”发送数据（并非即刻发送）
  - 当数据被发送至内核的接收缓冲区中，等待应用程序执行$recv()$接收数据
  - 若接收缓冲区中无数据
    - 将进程信息注册至等待队列中，进程休眠
    - 等待数据传入接收缓冲区中



HSPFuzzer: High-Speed Network Protocol Fuzzing With Connection Reuse

算法原理 TIPO

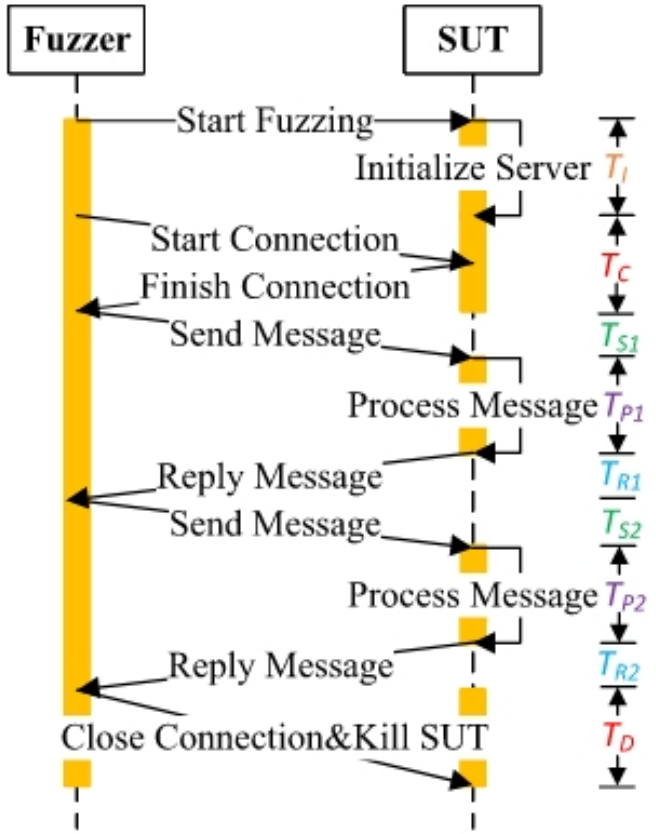
T	目标	实现高速协议模糊器，提高模糊测试吞吐量
I	输入	待测系统：12种网络协议实现的服务端（Dnsmasq、LightFTP、TinyDTLS、Kamailio、OpenSSL、OpenSSH、DCMTK、Live555、Mosquitto、ippsample、Redis、Exim） 初始种子：捕获的测试服务器与客户端的网络流量pcap文件
P	处理	1. 预处理过程：前缀报文识别 2. 模糊测试过程：连接重用，覆盖监控机制检测异常执行状态
O	输出	模糊测试吞吐量（执行模糊测试次数/s）、代码覆盖率、漏洞报告

P	问题	被测服务器重新启动、连接设置等开销导致模糊测试吞吐量低
C	条件	连接重用适用于基于TCP服务器
D	难点	1.前缀消息的准确识别 2.模糊器自动跟踪SUT消息处理状态，及时传递下一条消息
L	水平	IEEE Internet of Things Journal 2025 中科院1区

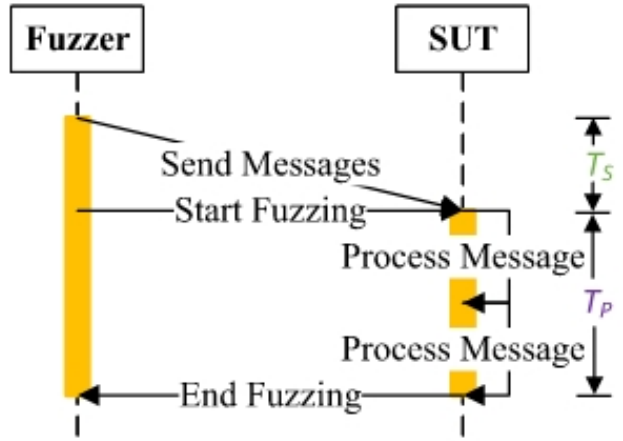
算法原理 协议模糊测试时间开销

- 处理单个种子的时间开销/μm
  - T_I： SUT初始化
  - T_C： 连接建立
  - T_S： 消息发送
  - T_P： SUT处理消息
  - T_R： Fuzzer接收消息
  - T_D： 连接关闭、终止测试

Subject	Fuzzer	T_I (Init. SUT)	T_C (Conn. Setup)	T_S (Send)	T_R (Reply)	T_P (SUT Process)	T_D (Close & Kill)
LightFTP	HNPFuzzer	15,678	873	35	27	112	1,573
	AFLNET	19,409	1,945	305	183	109	734
Live555	HNPFuzzer	2,400	225	56	29	783	623
	AFLNET	9,247	1,162	1,653	1,754	691	192
OpenSSL	HNPFuzzer	84,906	528	45	15	12,056	6,939
	AFLNET	70,667	10,381	718	333	11,436	5,893
DCMTK	HNPFuzzer	81,956	244	62	11	3,158	21,100
	AFLNET	45,045	1,176	567	341	2,758	1,940



同一连接中处理多条消息
降低连接建立相比于SUT
测试的成本

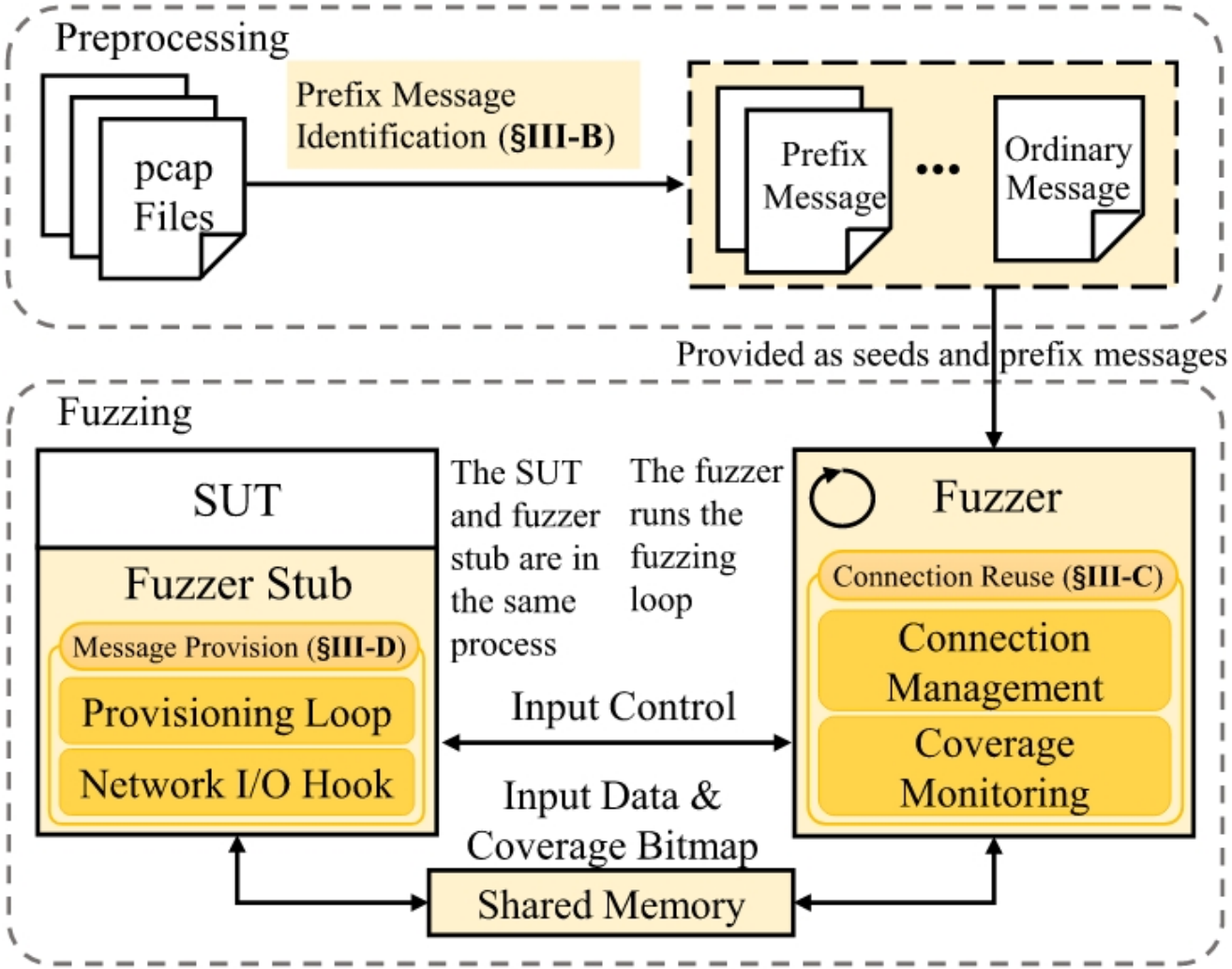


$$AFLNet - time = T_I + T_C + T_S + T_P + T_R + T_D$$

$$HSPFuzzer - time = (NT_I + MT_C + nT_S + nT_P + MT_D)/n$$
$$= \left(\frac{N}{n}\right)T_I + \left(\frac{M}{n}\right)(T_C + T_D) + T_S + T_P$$

算法原理 HSPFuzzer

- HSPFuzzer提高吞吐量方法
  - 预处理阶段
    - 前缀消息识别
  - 模糊测试阶段
    - 主模糊器Fuzzer
      - 连接重用
    - 模糊器存根Fuzzer Stub
      - 消息提供
      - 监控SUT（Server Under Test）状态
      - 确保及时传入消息
    - 使用共享内存相互通信



算法原理 HSPFuzzer 前缀消息识别

- 前缀消息
  - 前缀消息：必须在连接开始时发送
    - FTP协议中的身份验证$USER$, $PASS$
  - 普通消息：在身份验证后可无限次发送
    - FTP协议中目录/文件操作命令MKD , CWD等
  - 退出消息：明确终止连接
    - 结束会话或处理严重错误

```
1  220 LightFTP server v2.0a ready
2  USER foo
3  331 User foo OK. Password required
4  PASS foo
5  230 User logged in, proceed.
6  MKD demo
7  257 Directory created.
8  CWD demo
9  250 Requested file action okay, completed.
10 STOR test.txt
11 150 File status okay
12 226 Transfer complete
13 LIST
14 150 File status okay
15 226 Transfer complete
16 QUIT
17 221 Goodbye!
```

Protocol	Prefix Message	Ordinary Message	Quit Message	Server
FTP	{USER,PASS}	LIST,PWD,ABOR, ...	QUIT, over-length message	LightFTP v2.3.1, Pure-FTPd v1.0.52
			QUIT, messages of other protocols like GET, over-length messages	vsftpd v3.0.5, ProFTPD v1.3.8c
DTLS	{ClientHello, ClientHello[1]}	ClientKeyExchange, ChangeCipherSpec, Finished, ...	ALERT (fatal)[2], error messages (e.g., wrong fields)	TinyDTLS (8a9e048)
SIP	[REGISTER],[INVITE]	CREATE, JOIN, INVITE, ...	BYE, fatal error messages (i.e., wrong critical fields)	Kamailio v6.0.0
TLS	ClientHello	ClientKeyExchange, ChangeCipherSpec, ...	ALERT (fatal), error messages (e.g., wrong fields)	OpenSSL v3.4.0
SSH	{SSH_MSG_KEXINIT, SSH_MSG_KEX_ECDH_INIT, SSH_MSG_USERAUTH _REQUEST}, ...	SSH_MSG_CHANNEL_OPEN, SSH_MSG _CHANNEL_REQUEST, ...	SSH_MSG_DISCONNECT, unknown message types, error messages if in strict mode	OpenSSH v9_9_P1
DICOM	[A-ASSOCIATE-RQ]	C-STORE, C-FIND, ...	A-RELEASE-RQ, A-ABORT, fatal error messages (i.e., wrong critical fields)	DCMTK v3.6.9
RTSP	[DESCRIBE]	OPTIONS, SETUP, PLAY, ...	TEARDOWN, fatal error messages (i.e., wrong critical fields)	Live555 v2025.01.17

算法原理 HSPFuzzer 前缀消息识别

- 前缀消息识别$PrefixMsgs$
  - 若前缀消息不完整，则无法执行后续消息，导致SUT中的代码覆盖率不变

```
453 } else if (loggedin == 0) {
454   /* from this point, all commands need
         authentication */
455   addreply_noformat(530, MSG_NOT_LOGGED_IN);
456   goto wayout;
457 } else {
```

- 前缀消息与非前缀消息间是否拆分$NeedSplit$
  - 比较以下两类情况的代码覆盖率
    - 按顺序发送pcap文件所有消息$CovA$
    - 一次模糊测试处理所有消息$CovO$
- 非前缀消息间是否拆分$AllSplit$
- 使用llvm-cov计算代码覆盖率

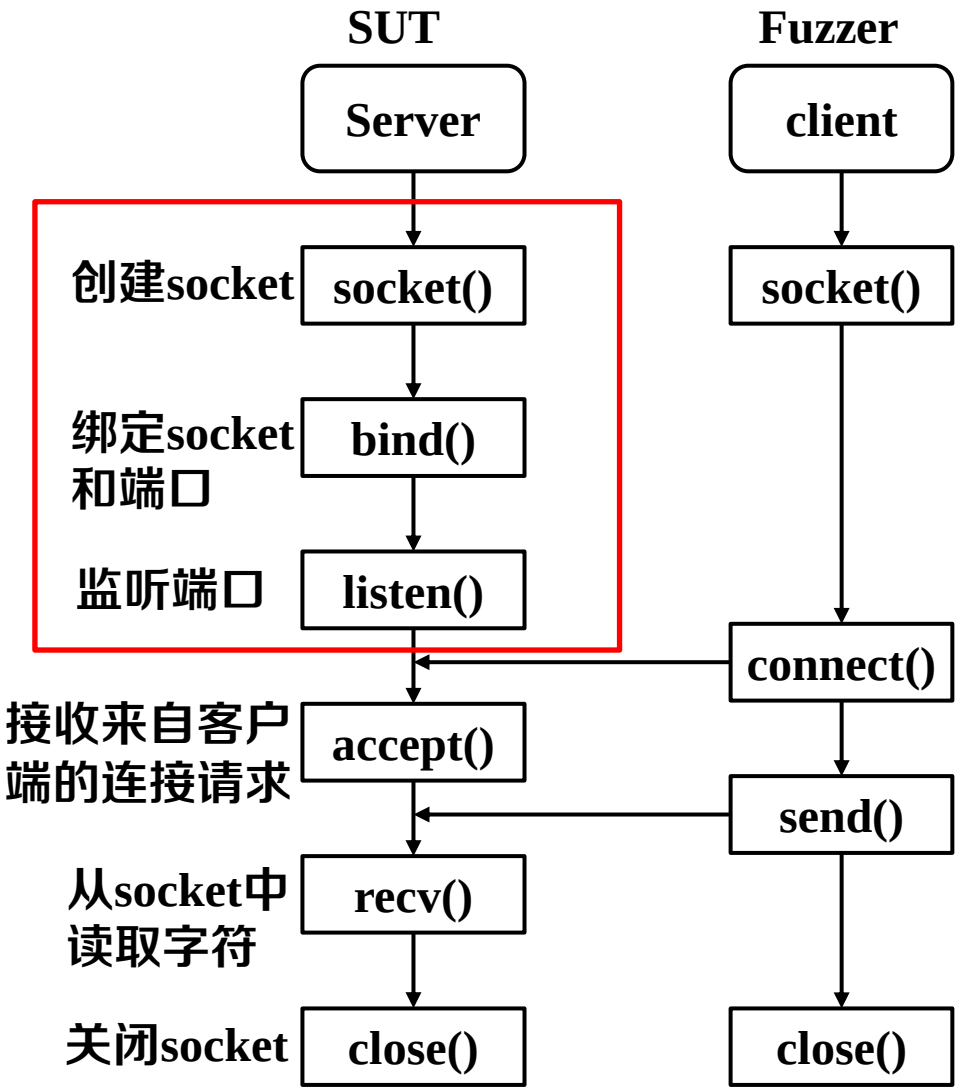
Algorithm 2 Prefix Message Identification

Input: *PcapFile*
Output: *PrefixMsgs*, *NeedSplit*, *AllSplit*

Msgs ← *PcapFile* // Get a list of messages $M_0 \dots M_{N-1}$
N ← *Msgs*.LENGTH // Assume $N \geq 2 + \tau$
$\{Cov(M_i) \mid i \in [0, N-1]\}$ ← GETCOVERAGE(*Msgs*)
// Determine PrefixMsgs
for $x \leftarrow 0$ **to** $N - 2 - \tau$ **do**
  LeftM ← *Msgs*.REMOVE(*Msgs*.GET(x))
  $\{CovT(M_i) \mid i \in [0, N-1] \wedge i \neq x\}$ ← GETCOVERAGE(*LeftM*)
  Num ← 0
  for $y \leftarrow x+1$ **to** $N-1$ **do**
    if DIFF($Cov(M_y)$,$CovT(M_y)$) $\leq \delta$ **then**
      Num ← *Num* + 1
    end if
  end for
  if *Num* $> \tau$ **then**
    break
  end if
end for
PrefixMsgs ← *Msgs*.SUBLIST(0, x)
// Determine NeedSplit and AllSplit
NeedSplit ← **false**, *AllSplit* ← **false**
MsgsInOne ← CONCATENATE/(*Msgs*)
CovO ← GETCOVERAGE(*MsgsInOne*)
CovA ← $\bigcup_0^{N-1} Cov(M_i)$
if DIFF(*CovA*,*CovO*) $> N\delta$ **then**
  NeedSplit ← **true**
  LeftMsgsInOne ← CONCATENATE(*Msgs*.SUBLIST(x, N))
  CovOL ← GETCOVERAGE(*PrefixMsgs*,*LeftMsgsInOne*)
  if DIFF(*CovA*,*CovOL*) $> N\delta$ **then**
    AllSplit ← **true**
  end if
end if

算法原理 HSPFuzzer 连接重用

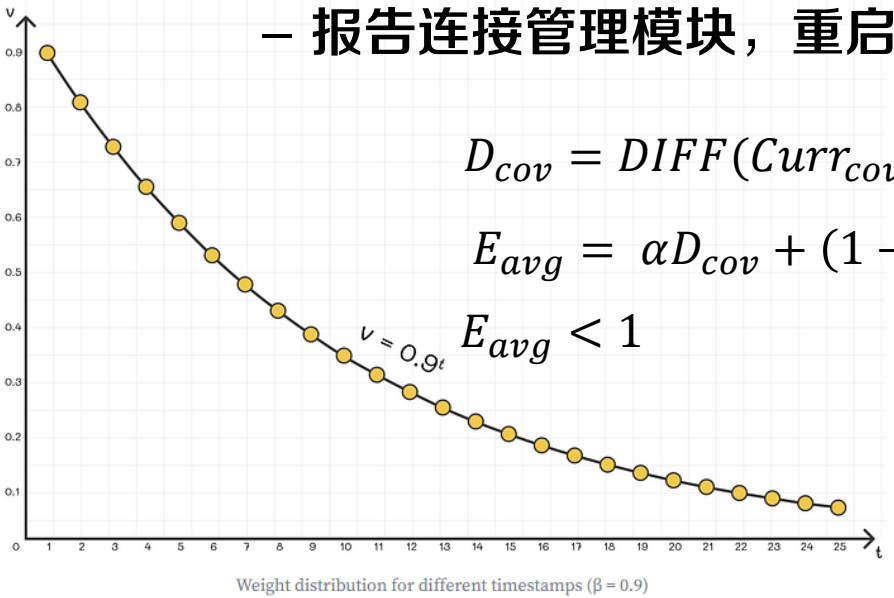
- Fuzzer：连接管理模块
  - 协调网络连接的建立和终止
    - 测试用例输入前检查模糊器Fuzzer是否与待测服务器SUT建立连接
    - Fuzzer Stub检测到连接已关闭或输入处理重复超时（例如，连续超时超过五次），则连接管理模块会将当前连接标记为关闭并启动一个新连接
  - 传输前缀数据包
  - 若因超时而频繁重新启动连接，重启SUT
- Fuzzer Stub
  - 连接重用：通过建立虚拟套接字，消除重新启动连接时的网络开销



算法原理 HSPFuzzer 连接重用

- **Fuzzer：覆盖率监控模块**
  - 基于指数加权移动平均算法（EWMA）
    - 越靠近当前时刻的数值，其权重越大，而较远的数值权重逐渐减小
  - 计算当前输入与先前输入的覆盖率差异D_{cov}与指数移动均值E_{avg}
    - $E_{avg} < 1$
      - 重复执行相同的操作而没有发现新的覆盖范围
  - 报告连接管理模块，重启新连接

一次连接中有多条消息进行模糊测试，
降低旧消息的权重，关注新消息的贡献

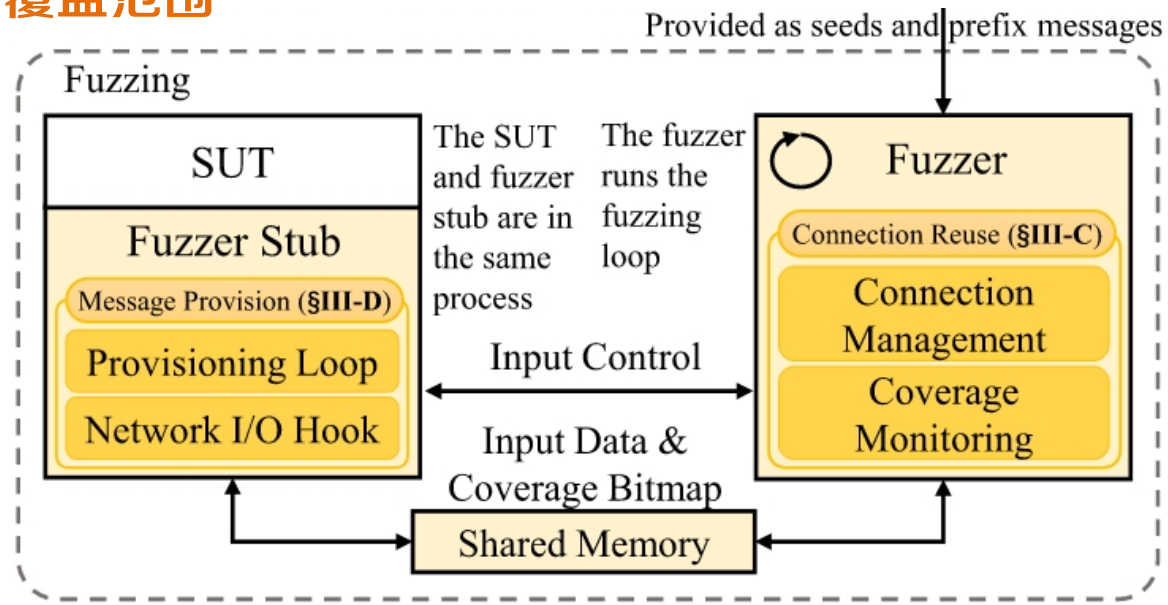


Weight distribution for different timestamps (β = 0.9)

$$D_{cov} = DIFF(Curr_{cov}, Prev_{cov})$$

$$E_{avg} = \alpha D_{cov} + (1-\alpha)E_{avg}$$

$$E_{avg} < 1$$

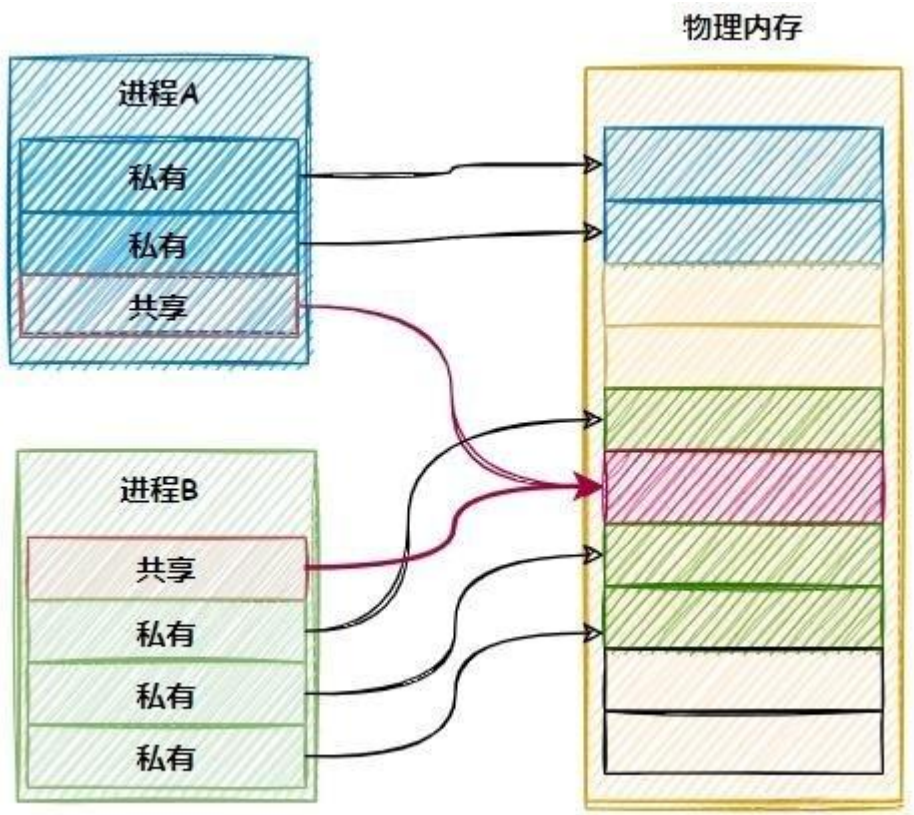


算法原理 HSPFuzzer 消息提供

- Fuzzer Stub 消息提供
  - 确保从Fuzzer接收的输入（消息）立即传送到 SUT 进行处理，减少延迟
- Network I/O Hook（ LD_PRELOAD ）
  - 创建虚拟套接字
  - 操纵套接字的读取和写入，使用共享内存优化速度
- Provisioning Loop
  - 确定SUT是否完成对输入中所有消息的处理，并能接收下一个输入
  - SUT尝试读取下一个输入，执行$recv()$
  - 显式连接终止
    - 表明SUT已收到退出消息并打算终止会话

信息监控

条件判断



实验设计　数据资源

- 数据资源

Server	Protocol	Source
Dnsmasq	DNS	profuzzbench
LightFTP	FTP	
TinyDTLS	DTLS	
Kamailio	SIP	
OpenSSL	TLS	
OpenSSH	SSH	
DCMTK	DICOM	
Live555	RTSP	
Exim	SMTP	
Mosquitto	MQTT	opensource
ippsample	IPP	
Redis	RESP	

- 对比方法
  - AFLNET（2020）
    - 首个协议状态感知的模糊测试方法
  - AFLNwe（2020）
    - 简单的网络支持AFL扩展，不考虑协议状态信息
  - AFL++&desock（2020）
  - SnapFuzz（2022）
    - 将异步网络通信转换为同步通信
  - HNPFuzzer（2024）
    - 使用共享内存代替socket套接字传递消息
- 评价指标
  - 模糊测试吞吐量（每秒执行模糊测试次数）
  - 代码覆盖率

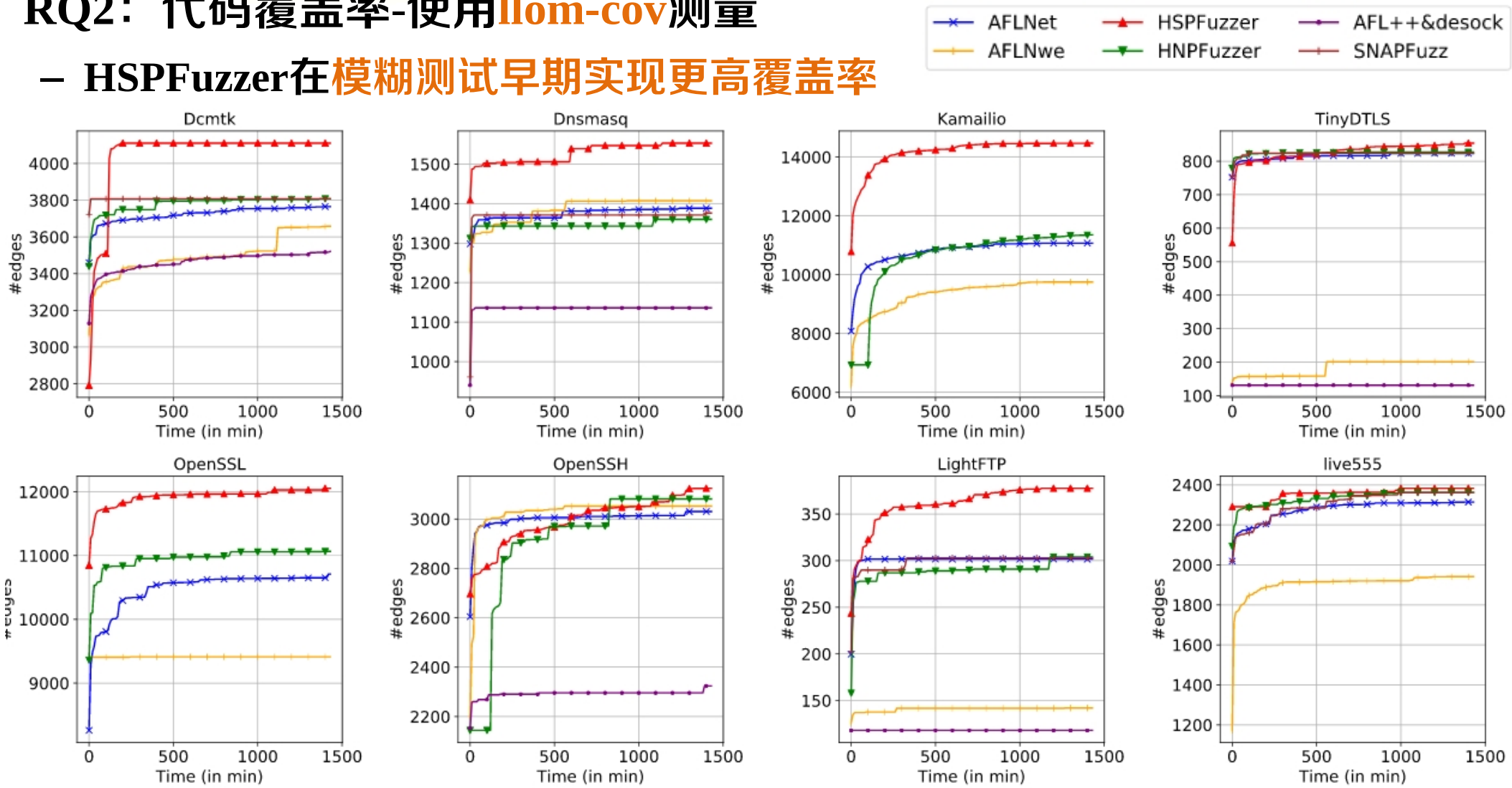
实验结果　对比实验

- **RQ1：模糊测试吞吐量**
  - 连接重用与高效的消息提供机制可减少开销
  - 吞吐量相较于**AFLNET提高106206.32%**
  - **平均吞吐量为4473.05次/秒**

Server	AFLNET	AFLNWE	SNAPFuzz	HNPFuzzer	AFL++&desock	HSPFuzzer
Dnsmasq	6.82	27.49 (+303.08%)	62.39(+814.81%)	32.55 (+377.27%)	46.66 (+584.16%)	10359.15 (+151793.70%)
LightFTP	5.12	32.04 (+525.78%)	31.20 (+509.38%)	31.43 (+513.87%)	22.70 (+343.36%)	12058.14 (+235410.55%)
TinyDTLS	2.12	14.08 (+564.15%)	49.50 (+2234.91%)	219.88 (+10271.70%)	43.01 (+1928.77%)	11154.47 (+526054.25%)
Kamailio	4.61	5.80 (+25.81%)	-	8.43 (+82.86%)	-	6028.45 (+130668.98%)
OpenSSL	3.41	14.96 (+338.71%)	-	8.69 (+154.84%)	-	3668.30 (+107474.78%)
OpenSSH	19.12	22.02 (+15.17%)	-	27.7 (+44.87%)	3.2 (-83.26%)	29.65 (+55.07%)
DCMTK	18.42	15.83 (-14.06%)	24.74 (+34.31%)	6.97 (-62.16%)	17.84 (-41.15%)	792.13 (+4200.38%)
Live555	12.16	30.46 (+150.49%)	50.60 (+316.12%)	14.07 (+15.71%)	-	824.40 (+6679.61%)
Mosquitto	8.45	4.58 (-45.80%)	-	-	31.46 (+272.31%)	3683.51 (+43491.83%)
ippsample	7.31	9.09 (+24.35%)	-	41.52 (+467.99%)	37.20 (+408.89)	4005.61 (+54696.31%)
Redis	7.34	10.25 (+32.43%)	-	-	33.81 (+336.82%)	1065.31 (+13663.70%)
Exim	1.95	3.81 (+95.83%)	-	7.16 (+267.18%)	-	7.54 (+286.67%)
Average		**167.96%**	**781.90%**	**1213.41%**	**405.54%**	**106206.32%**

实验结果 对比实验

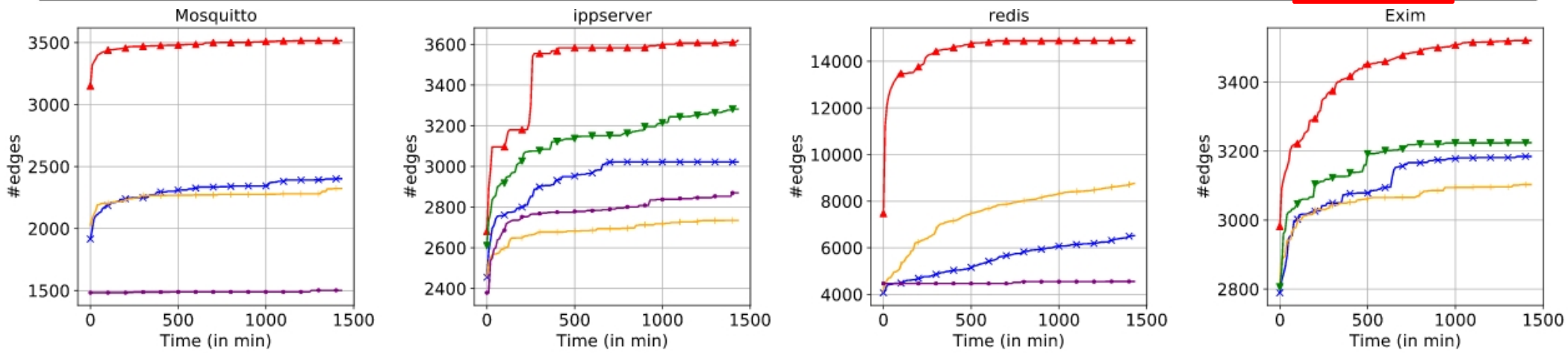
- RQ2：代码覆盖率-使用llom-cov测量
  - HSPFuzzer在模糊测试早期实现更高覆盖率



实验结果　对比实验

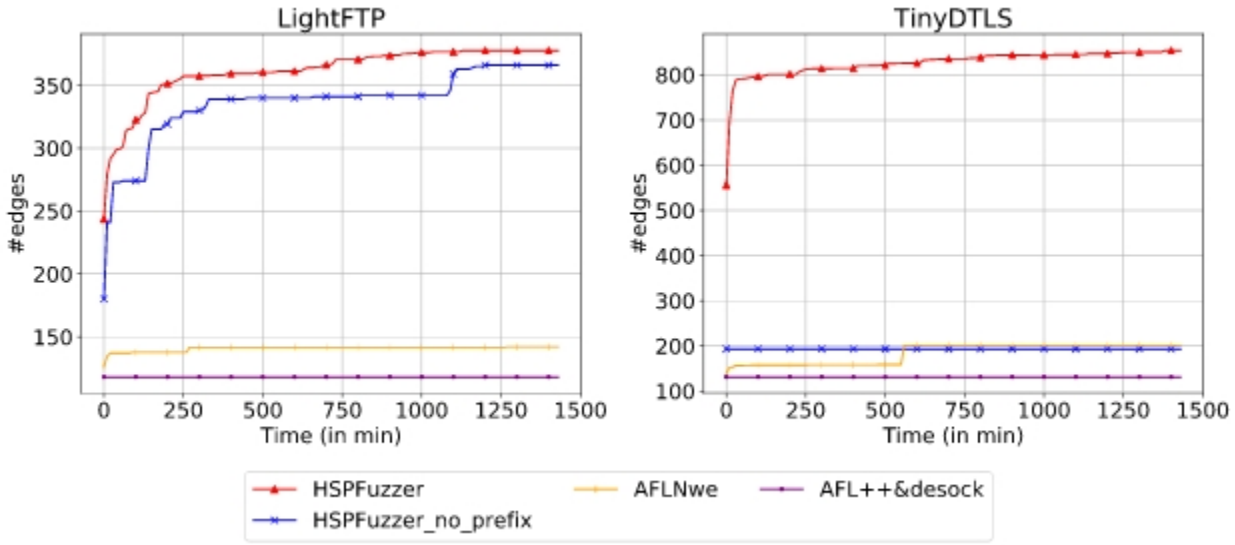
- **RQ2：代码覆盖率-使用llom-cov测量**

Server	AFLNET	AFLNWE	SNAPFuzz	HNPFuzzer	AFL++&desock	HSPFuzzer	p-value
Dnsmasq	1388.00	1407.00 (+1.37%)	1376.20 (-0.85%)	1381.90(-0.44%)	1136.30 (-18.13%)	1553.00 (+11.89%)	0.012
LightFTP	301.70	142.00 (-52.93%)	303.00 (+0.43%)	304.50 (+0.93%)	118.20 (-60.82%)	377.70 (+25.19%)	0.008
TinyDTLS	823.70	201.30 (-75.56%)	824.30 (+0.07%)	827.00 (+0.40%)	131.50 (-84.04%)	854.00 (+3.68%)	0.015
Kamailio	11075.00	9753.30 (-11.93%)	-	11358.80 (+2.56%)	-	14474.30 (+30.69%)	<0.001
OpenSSL	10711.30	9411.00 (-12.14%)	-	11066.70 (+3.32%)	-	12062.10 (+12.61%)	0.004
OpenSSH	3047.30	3052.30 (+0.16%)	-	3081.30 (+1.12%)	2413.30 (-20.81%)	3124.70 (+2.54%)	0.029
DCMTK	3765.50	3658.50 (-2.84%)	3805.00 (+1.05%)	3808.20 (+1.13%)	3521.30 (-6.49%)	4111.00 (+9.18%)	0.011
Live555	2314.00	1941.00 (-16.12%)	2364.70 (+2.19%)	2362.50 (+2.10%)	-	2383.00 (+2.98%)	0.017
Mosquitto	2403.40	2323.00 (-3.35%)	-	-	1564.20 (-34.92%)	3530.40 (+46.89%)	0.003
ippsample	3017.20	2735.00 (-9.35%)	-	3286.40 (+8.92%)	2892.30 (-4.14%)	3619.10 (+19.95%)	0.009
Redis	6537.80	8789.40 (+34.44%)	-	-	4556.00 (-30.31%)	14893.80 (+127.81%)	<0.001
Exim	3184.30	3103.00 (-2.55%)	-	3224.40 (+1.26%)	-	3423.70 (+7.80%)	0.002
Average		**-12.57%**	**+0.58%**	**+2.13%**	**-32.46%**	**+25.10%**	

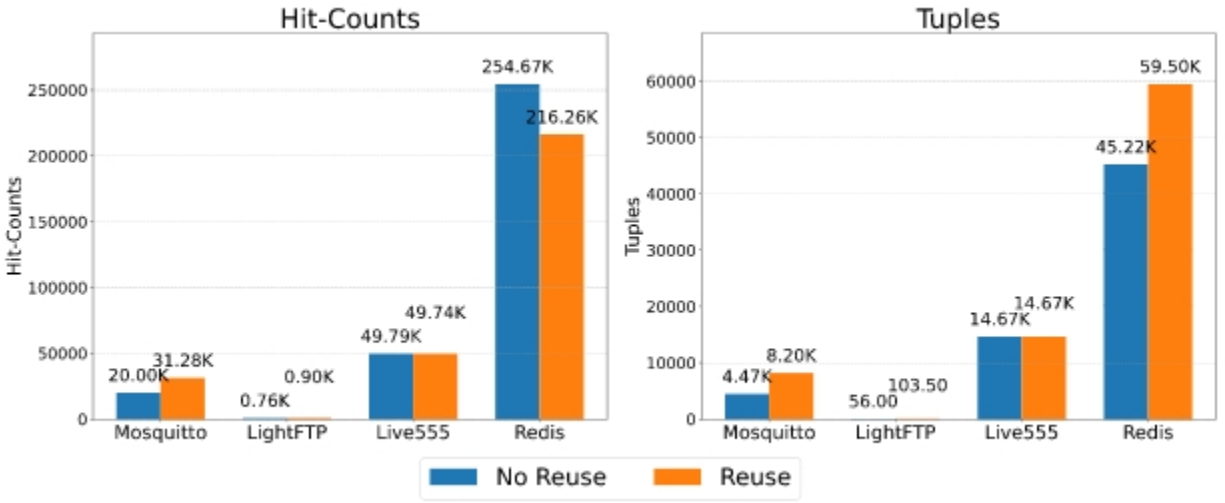


实验结果 对比实验

- RQ3：连接重用机制
  - 前缀消息识别
  - 连接重用不影响对SUT执行路径的探索
    - 使用命中计数和基本块覆盖率评估



- RQ4：Bug发现数量与效率
  - HSPFuzzer可在前1h内发现大部分问题



Subject	Vulnerability	AFLNet	AFLNwe	SnapFuzz	HNPFuzzer	AFL++	HSPFuzzer
Live555	CVE-2018-4013	11m50s	20m39s	1m42s	10m55s	✗	1m1s
	CVE-2021-38381	8m57s	34m31s	15m35s	17m0s	✗	7m21s
	CVE-2021-38382	13m5s	26m16s	12m35s	12m52s	✗	12m20s
	CVE-2021-39282	18m45s	44m14s	14m54s	12m15s	✗	19m49s
TinyDTLS	Bug#544819	<1m	<1m	<1m	<1m	✗	<1m
	buffer-overflow2	1m53s	<1m	<1m	<1m	✗	<1m
	buffer-overflow3	5m51s	7m36s	3m11s	<1m	✗	<1m
	assertion fail	✗	✗	✗	14m32s	✗	152m47s
DCMTK	Bug#942	105m57s	156m32s	✗	✗	✗	31m5s
	segfault	181m20s	174m53s	14m32s	10m37s	1000m 30s	37m25s
	SEGV	213m2s	150m43s	✗	✗	✗	51m58s
	CVE-2021-41690	✗	✗	✗	✗	✗	18m
	memory leak	✗	✗	✗	✗	✗	11m34s
Redis	CVE-2024-31227	✗	✗	✗	✗	✗	4m51s
Number	14	10	10	8	9	1	14

算法总结

- 算法贡献
  - 使用连接重用降低模糊测试开销
    - 同一连接内处理多条消息：确定并发送前缀消息后，可在此连接内发送多条普通消息，避免为每条消息重复建立连接
    - 建立虚拟socket套接字：SUT 启动新连接时，Fuzzer与虚拟套接字进行互联，减少每次连接建立前的资源占用
  - Fuzzer与Fuzzer stub之间使用共享内存通信
    - 减少冗余的读写操作延时
  - 高效消息提供机制
    - 监控SUT执行状态，确保下一条输入及时传入
- 算法不足
  - 正交于种子选取、变异策略等方向，可尝试有效结合

No Peer, no Cry: Network Application Fuzzing via Fault Injection

算法原理 TIPO

T	目标	对客户端/服务端源码进行故障注入生成错误消息，进行协议模糊测试
I	输入	故障注入端（客户端/服务端）源代码 目标端（服务端/客户端）源代码
P	处理	1.故障注入：随机位置注入随机4种故障类型 2.模糊测试：代码覆盖率指导
O	输出	代码覆盖率、目标端（服务端/客户端）异常行为，漏洞报告

P	问题	现有方法关注协议空间，变异后的种子难以通过加密、交换签名等完整校验，导致通信提前终止
C	条件	可访问客户端/服务端源代码，可监控目标端状态
D	难点	故障注入后需保障有效通信
L	水平	ACM Conference on Computer and Communications Security 2024 CCF A

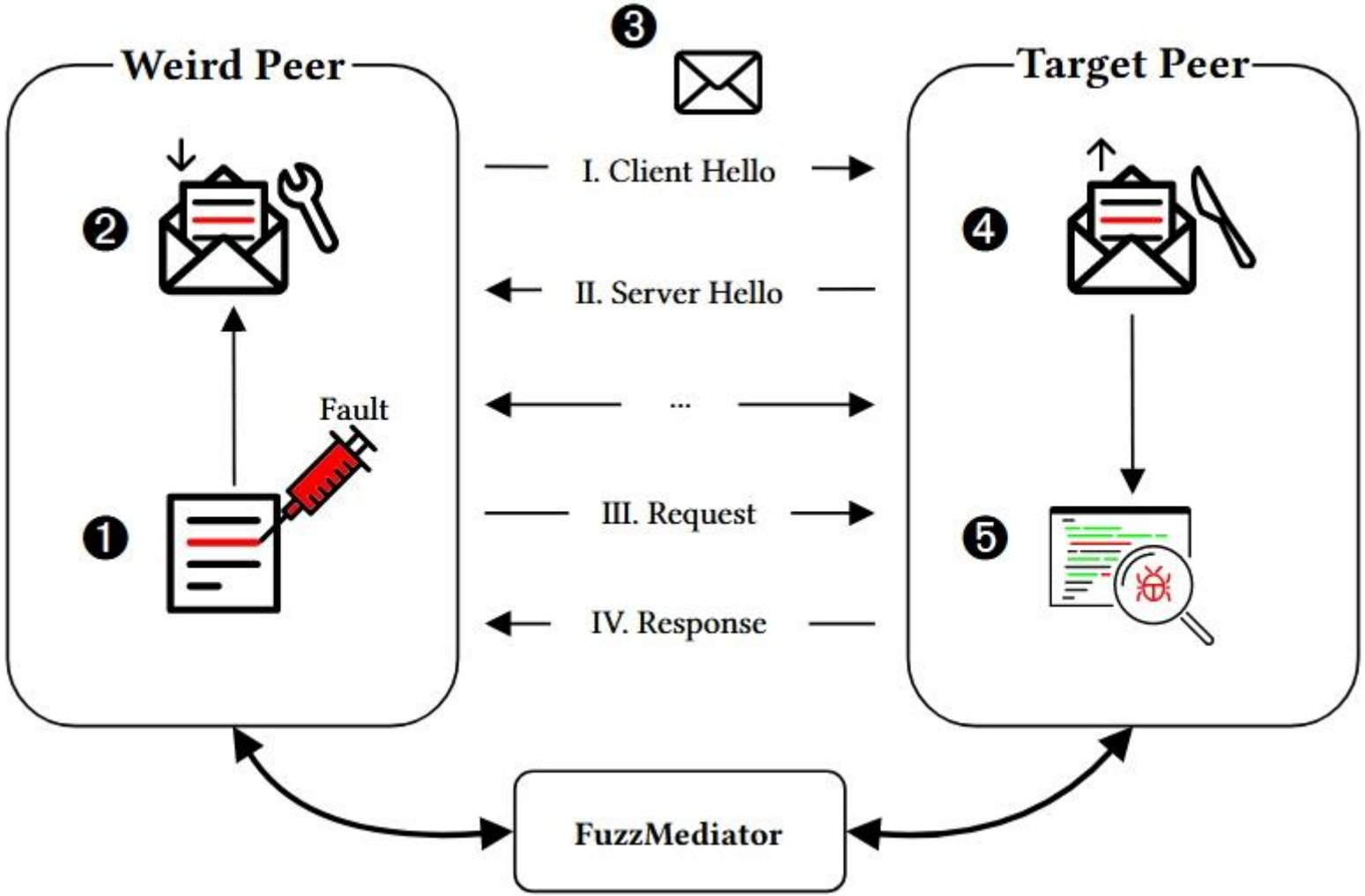
算法原理 创新说明

- 协议模糊测试难点
  - $C1$ 会话状态
    - 客户端和服务端状态同步问题
    - 会话中的临时标识符
  - $C2$ 完整性校验与加密
    - 通信过程中常用编码、校验和、加密方法，保护消息
  - $C3$ 网络应用配置
    - 考虑发送模糊测试输入的时间
    - 考虑模糊测试中会话终止条件

- 现有方法未能解决$C1$ $C2$
  - 基于重放的方法（replay-based）
    - 模糊器扮演客户端或服务器，对先前捕获到的通信流量进行变异
  - 基于中间人攻击方法(Man-in-the-middle, MiTM)
    - 模糊器充当通信双方之间的代理，对正常通信流量进行变异或注入
- Fuzztruction-Net
  - 对客户端/服务器进行故障注入
  - 在通信系统中引入模糊器，不破坏通信双方与通信通道

算法原理 Fuzztruction-Net

- 核心算法原理
  - 故障注入
    - 故障类型，故障位置
    - 代码覆盖率引导
  - 模糊测试
  - 网络配置
    - 服务器准备好接收模糊测试输入
    - 一次模糊测试结束后使用终止信号关闭服务器



算法原理 Fuzztruction-Net 故障注入

- 利用故障注入破坏消息内容，不影响消息的后续处理

- 故障注入类型
  - 修改值
    - 加载/存储：修改从内存加载/存储到内存的值
  - 修改控制流
    - 选择：改变执行的case语句
    - 条件：反转条件或移动分支
    - 调用：更改调用地址或跳过

- 故障注入位置
  - 随机引入故障，修剪无效故障
  - 基于代码覆盖率引导，降低对“产生进程提前终止”等故障的调用

如何用模糊测试的思想发现在哪些位置，注入什么故障

算法原理 Fuzztruction-Net 模糊测试

- 模糊测试循环流程
  - “初始种子”生成
    - 队列条目$[(loc, stream), \ldots]$，其中loc表示错误注入位置，$stream$表示该位置应用的字节流（改变值、条件或调用目标）
    - 根据目标端覆盖率是否更新，选择将一元组$(loc, stream)$加入队列
  - “种子”变异
    - 对$stream$应用havoc变异策略进行变异
    - 队列条目拼接：已知+已知，已知+未知
  - 执行变异后“种子”（故障注入）
  - 运行故障注入后的客户端/服务端
  - 收集执行覆盖率

Type	Meaning	Mutator
bitflip	Flip a bit at a random position.	`bitflip 1`

interesting values	Set bytes with hard-coded interesting values.	`interest 8`
		`interest 16`
		`interest 32`

arithmetic increase	Perform addition operations.	`addition 8`
		`addition 16`
		`addition 32`

arithmetic decrease	Perform subtraction operations.	`decrease 8`
		`decrease 16`
		`decrease 32`

random value	Randomly set a byte to a random value.	`random byte`
delete bytes	Randomly delete consecutive bytes.	`delete chunk bytes`
clone/insert bytes	Clone bytes in 75%, otherwise insert a block of constant bytes.	`clone/insert chunk bytes`
overwrite bytes	Randomly overwrite the selected consecutive bytes.	`overwrite chunk bytes`

实验设计　数据资源

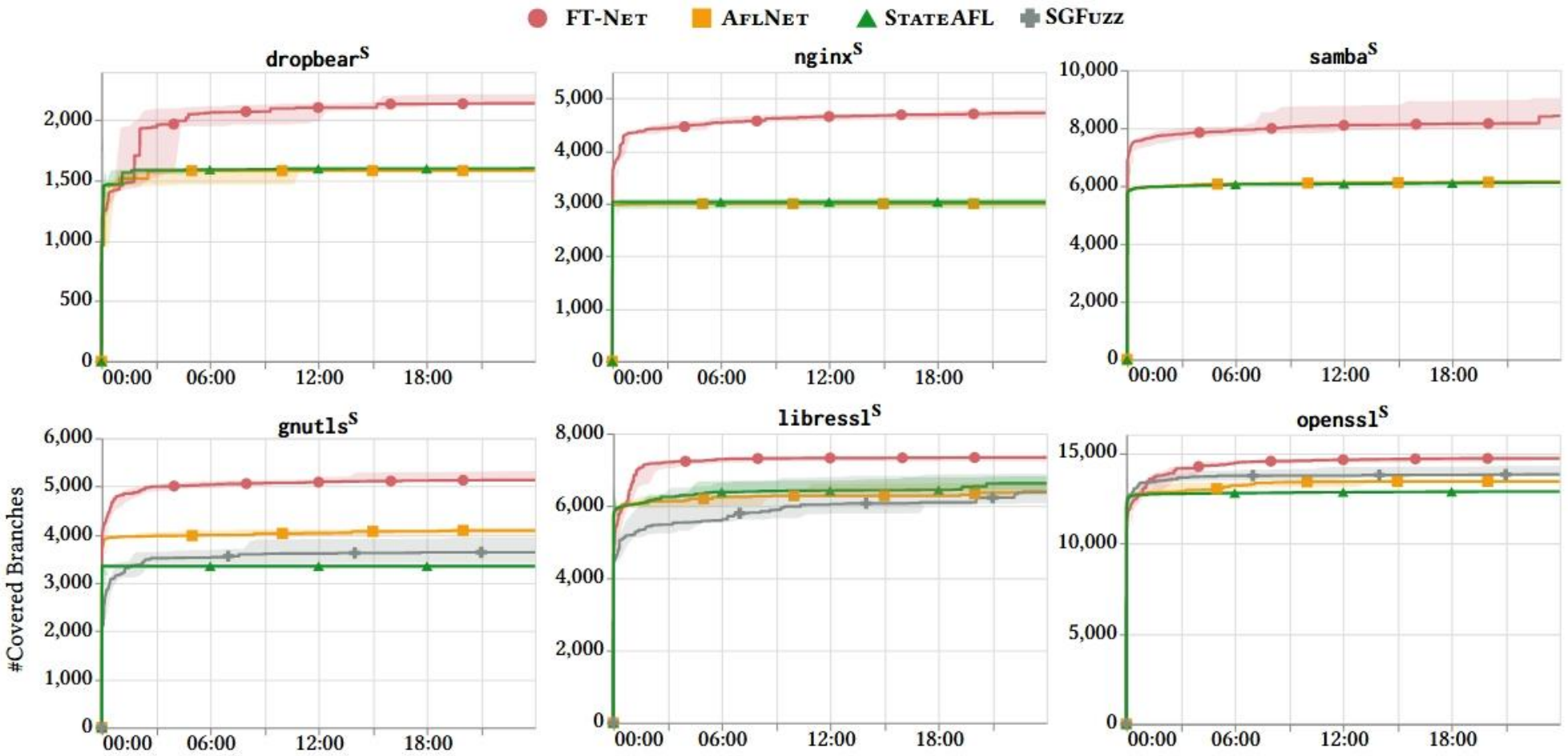
- 数据资源

Target	Protocol	Version
dropbear ^{S}	DNS	9925b00
dcmtk ^{S}	FTP	1549d8c
gnutls ^{S}	DTLS	E840a07
libressl ^{S}	SIP	Fbb21ed
live555 ^{S}	TLS	2023.06.14
mosquitto ^{S}	SSH	3923526
nginx ^{S}	DICOM	6b1bb99
openssl ^{S}	RTSP	7b649c7
samba ^{S}	SMTP	95474d8

- 对比方法
  - AFLNet（2020）
    - 首个协议状态感知的模糊测试方法
  - SGFuzz（2022）
    - 在源代码中使用枚举变量来捕获状态转换
  - STATEAFL（2022）
    - 通过分析持久性内存区域的快照来推断服务器状态
- 评价指标
  - 代码覆盖率、漏洞发现数量

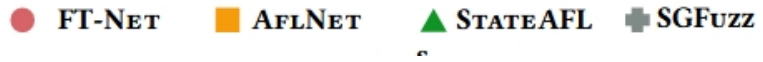
实验结果　对比实验

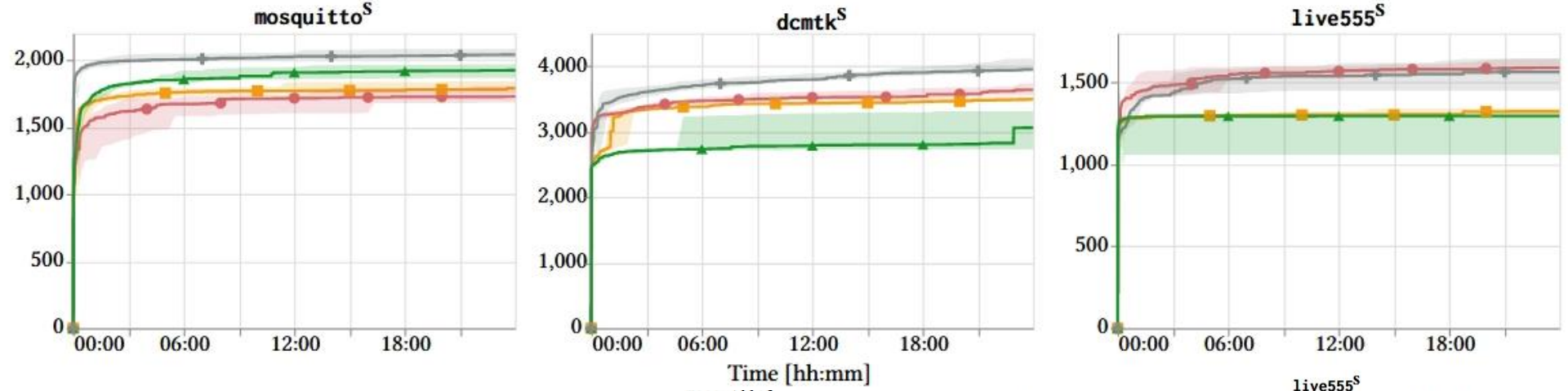
- 代码覆盖率-使用llom-cov测量
  - 优势分析：变异后的种子能够保证会话状态、通过完整性校验等情况



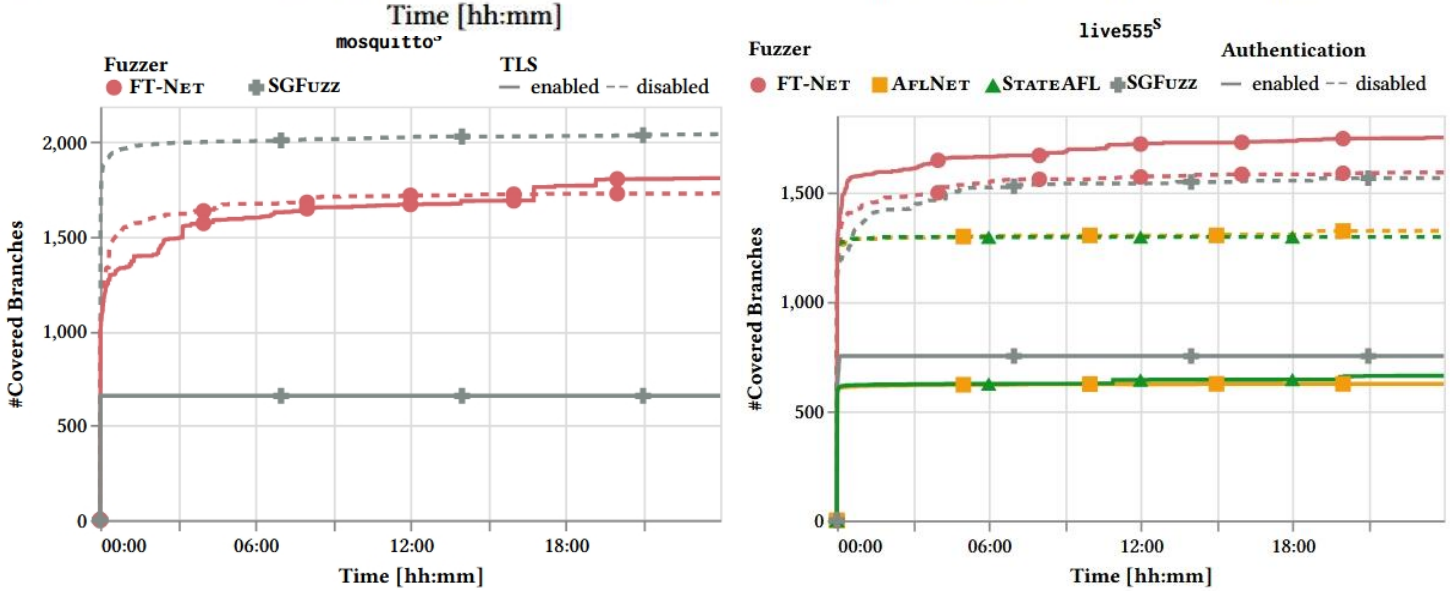
实验结果　对比实验

- 代码覆盖率-使用llom-cov测量
  - MQTT等协议格式简单，相较于对客户端进行故障注入，直接进行位级突变更有效





  - 更适用于实际情况
    - TLS加密
    - HTTP身份验证



实验结果　漏洞发现

Target	Weird Peer	Status	Type	Description
nginx^S	ngtcp2^C	*fixed*	Heap OOB read	ngx_quic_parse_transport_param uses user-controlled length value
nginx^S	ngtcp2^C	CVE-2024-32760	Heap OOB write	Duplication of an entry in the HTTP3 dynamic table, while the table is too small
nginx^S	ngtcp2^C	CVE-2024-31079	Heap UAF	During an infinite recursion, the recurringly accessed memory pool is freed
nginx^S	ngtcp2^C	CVE-2024-3416	Info. leak.	QUIC packets can cause worker processes to leak previously freed memory
nginx^S	ngtcp2^C	CVE-2024-35200	Nullptr deref	Access to a variable holding the user-agent header value is unexpectedly null
curl^C	nginx^S	*fixed*	Nullptr deref	ngtcp2_ksl_begin in the ngtcp2 library tries to accesses the head of a list that is null
curl^C	nginx^S	*fixed*	Assertion	In the nghttp3 library, an assertion checking if enough data is remaining is triggered
apache^S	curl^C	*reported*	Heap UAF	apr_file_write attempts to write error log after freeing its path
apache^S	curl^C	*reported*	Assertion	While adding a key-value pair into an APR table, an assertion is triggered
openssh^C	dropbear^S	*fixed*	Heap UAF	The asynchronous callback verify_host_key accesses the previously freed host key
dropbear^C	dropbear^S	*fixed* (#285)	Assertion	signkey_type_from_signature tries to cast an invalid integer to an enum
gnutls^S	gnutls^C	*fixed* (#1529)	Nullptr deref	During a retry-handshake, _gnutls_cipher_auth dereferences a null pointer
gnutls^S	gnutls^C	*fixed* (#1534)	Nullptr deref	_gnutls_figure_common_ciphersuite dereferences a null ptr if a PSK cipher is used
libressl^C	libressl^S	*fixed* (#1037)	Nullptr deref	Attempting to print key material in ssl_print_tmp_key after it has been freed
ngtcp2^S	ngtcp2^C	*fixed* (#1529)	Assertion	The wolfssl library returns a TLS session using the unauthenticated CTR AES mode, even though the CCM mode was negotiated
ngtcp2^S	ngtcp2^C	*fixed* (#7406)	Heap OOB write	In wolfSSL_read_early_data, header bytes are written to a insufficiently sized buffer
pjsip^S	pjsip^C	*fixed*	Heap OOB read	A timer object is deleted before it expires, causing an out-of-bound read
pjsip^S	pjsip^C	*fixed*	FP Exception	Proposing a clock rate of 0 for an audio stream causes a division by zero
pjsip^S	pjsip^C	*fixed*	Heap OOB write	Dumped source address of an SDP message is copied into too small buffer
dcmtk^S	dcmtk^C	CVE-2024-34508	Nullptr deref	checkAndProcessSTORERequest accesses the value of an element pointing to null
dcmtk^S	dcmtk^C	CVE-2024-34509	Nullptr deref	DIMSE_parseCmdObject attempts to access a string that points to null
live555^S	live555^C	*reported*	Heap UAF	sendDataOverTCP sends MP3 bytes after freeing the RTSPClientConnection
live555^S	live555^C	*reported*	Heap UAF	During execution of handleHTTPCmd_TunnelingPOST, the processed input is freed

算法总结

- 算法贡献
  - 提出基于故障注入的协议模糊测试方法
    - 在通信系统中引入模糊器，不破坏通信双方与通信信道
    - 更贴合实际应用场景
  - 现有方法中，唯一能对客户端以及服务端进行模糊测试
- 算法不足
  - 难以应对即时通讯场景
    - 不能同时获得客户端与服务端的源码以及响应状态
  - 模糊测试开销问题并未考虑
    - 服务端与客户端之间建立连接的时间开销问题

特点总结与未来展望

特点总结与未来展望

- 特点总结
  - HSPFuzzer & Fuzztruction-Net
    - 提高模糊测试吞吐量
      - 使用连接重用机制降低模糊测试开销
      - 利用共享内存通信，减少额外读写耗时
    - 考虑现实应用场景
      - 考虑TLS加密、完整性校验等情况，避免通信进程过早终止
      - 利用故障注入方法对产生消息的源端进行变异
- 未来展望
  - 未知协议模糊测试，即时通讯情况
  - 同时实现高吞吐量与更贴切现实应用场景的协议模糊器

参考文献

[1] Zhang X, Zhang C, Li X, et al. A survey of protocol fuzzing[J]. ACM Computing Surveys, 2024, 57(2): 1-36.

[2] Bars N, Schloegel M, Schiller N, et al. No Peer, no Cry: Network Application Fuzzing via Fault Injection. Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security[C]. New York, NY : ACM, 2024: 750-764.

[3] Xia Z, Zeng Y, Song X, et al. HSPFuzzer: High-Speed Network Protocol Fuzzing With Connection Reuse[J]. IEEE Internet of Things Journal, 2025, 12(19): 40779-40792.

道德经

知人者智，自知者明。胜人者有力，自胜者强。知足者富。强行者有志。不失其所者久。死而不亡者，寿。

谢谢！