Performance Analysis of MYSEA

Ping, Chua Kai

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

PERFORMANCE ANALYSIS OF MYSEA

by

Chua Kai Ping

September 2012

Thesis Co-Advisors: Mark A. Gondree
Cynthia E. Irvine

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
19–9–2012	Master's Thesis	Sep 2011 - Sep 2012

4. TITLE AND SUBTITLE

PERFORMANCE ANALYSIS OF MYSEA

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Chua Kai Ping

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number: N/A

14. ABSTRACT

The Monterey Security Architecture (MYSEA) provides trusted security services, allowing users to access information at different sensitivity levels at the same time. The MYSEA server enforces a mandatory access control policy to ensure that users can only access data for which they are authorized. We would like to know the consequences of the MYSEA design on the performance of the MYSEA system. In particular, have the MYSEA trusted processes introduced any design bottlenecks into the system?

The objective of this thesis is to analyze the performance of selected aspects of MYSEA and, when applicable, identify system performance bottlenecks. In the absence of bottlenecks, our secure system performance study can be interpreted as characterizing the "cost of security" in a multilevel security context. We analyze the overhead associated with MYSEA by targeting and benchmarking its components and services. We deployed the netperf tool as a MYSEA service, to observe costs associated with IPSec, the MYSEA trusted proxy and communication among servers in the MYSEA Federation. Our benchmark tests provided useful insights to the performance overhead introduced by MYSEA's design and highlighted the cost of security of selected aspects in MYSEA.

15. SUBJECT TERMS

Multilevel security, cyber supporting, benchmark, performance penalties, overhead, trusted computing, MYSEA

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	UU	168	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

PERFORMANCE ANALYSIS OF MYSEA

Chua Kai Ping
Civilian, Defence Science & Technology Agency, Singapore
B.Eng., National University of Singapore, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2012

Author: Chua Kai Ping

Approved by: Mark A. Gondree
Thesis Co-Advisor

Cynthia E. Irvine
Thesis Co-Advisor

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Monterey Security Architecture (MYSEA) provides trusted security services, allowing users to access information at different sensitivity levels at the same time. The MYSEA server enforces a mandatory access control policy to ensure that users can only access data for which they are authorized. We would like to know the consequences of the MYSEA design on the performance of the MYSEA system. In particular, have the MYSEA trusted processes introduced any design bottlenecks into the system?

The objective of this thesis is to analyze the performance of selected aspects of MYSEA and, when applicable, identify system performance bottlenecks. In the absence of bottlenecks, our secure system performance study can be interpreted as characterizing the "cost of security" in a multilevel security context. We analyze the overhead associated with MYSEA by targeting and benchmarking its components and services. We deployed the netperf tool as a MYSEA service, to observe costs associated with IPSec, the MYSEA trusted proxy and communication among servers in the MYSEA Federation. Our benchmark tests provided useful insights to the performance overhead introduced by MYSEA's design and highlighted the cost of security of selected aspects in MYSEA.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Purpose of Study 1
1.3 Thesis Organization 2

2 Monterey Security Architecture Overview **5**
2.1 MYSEA Design Overview 6
2.2 MYSEA Concept of Operations 6
2.3 Overview of MYSEA Components and Services 8
2.4 MYSEA Security Policies 10
2.5 Current MYSEA Prototype 11

3 Objectives and High-Level Analysis **13**
3.1 Benchmarking Methodology 15
3.2 Performance Indicators 16
3.3 Benchmarking Tools for Use in MYSEA 17

4 Methodology **21**
4.1 Tools and Instrumentation 21
4.2 Test Environment 23
4.3 Test Configuration 24
4.4 Types of Tests 27
4.5 Test Plan 30

5 Testing and Analysis **33**
5.1 Overview 33

5.2 Summary of Results 34

6 Conclusion **41**
6.1 Recommendations for Future Work 41

Appendices **42**

A Netserver Modifications **43**

B Installing and Running the Benchmark Tool **47**
B.1 Installing Netserver 47
B.2 Installing Netperf 48
B.3 Execute benchmark tests 48

C Overview of Results **51**

D Results of IPSec Comparison **57**

E Results of MYSEA Comparison **89**

F Results of APS and TPS on different servers **121**

G UDP Discussion **137**

List of References **141**

Initial Distribution List **145**

List of Figures

Figure 2.1 Major Components in MYSEA From [1] 7

Figure 3.1 Process flow of a client making a service request to MYSEA 13

Figure 4.1 The MYSEA benchmark test environment (network configuration and hardware). 23

Figure 4.2 Figure showing the comparisons of different test configurations. Test cases considered in this thesis are underlined 26

Figure 5.1 Boxplot showing the summary of TCP_STREAM performance for the six setups 38

Figure 5.2 Boxplot showing the summary of TCP_CC socket performance for the six setups 39

Figure 5.3 Boxplot showing the summary of TCP_RR request and response performance for the six setups 40

Figure A.1 Insertion of MYSEA header with flags to compile for MYSEA support 43

Figure A.2 Addition of headers to source code in file: netserver.c 43

Figure A.3 Addition of functions to source code in file: netserver.c, function: main 44

Figure A.4 Modifications to source code in file: netserver.c, function: main 45

Figure A.5 Modifications to source code in file: netserver.c, function: main 46

Figure C.1 Boxplots of TCP_STREAM Tests of all scenarios 52

Figure C.2 Boxplots of UDP_STREAM Tests of all scenarios 53

Figure C.3 Boxplots of TCP_CC Tests of all scenarios 54

Figure C.4 Boxplots of TCP_CRR Tests of all scenarios 55

Figure C.5 Boxplots of TCP_RR Tests of all scenarios 56

Figure D.1 Boxplots comparison of IPSec using TCP_STREAM Tests 58

Figure D.2 Histogram comparison of IPSec using TCP_STREAM Tests 59

Figure D.3 Scatterplot comparison of IPSec using TCP_STREAM Tests 60

Figure D.4 Boxplots comparison of the performance of IPSec using UDP_STREAM Tests 61

Figure D.5 Histogram comparison of the performance of IPSec using UDP_STREAM Tests 62

Figure D.6 Scatterplot comparison of the performance of IPSec using UDP_STREAM Tests 63

Figure D.7 Boxplots comparison of the performance of IPSec using TCP_CC Tests 64

Figure D.8 Histogram comparison of the performance of IPSec using TCP_CC Tests 65

Figure D.9 Scatterplot comparison of the performance of IPSec using TCP_CC Tests 66

Figure D.10 Boxplots comparison of the performance of IPSec using TCP_CRR Tests 67

Figure D.11 Histogram comparison of the performance of IPSec using TCP_CRR Tests 68

Figure D.12 Scatterplot comparison of the performance of IPSec using TCP_CRR Tests 69

Figure D.13 Boxplots comparison of the performance of IPSec using TCP_RR Tests 70

Figure D.14 Histogram comparison of the performance of IPSec using TCP_RR Tests 71

Figure D.15 Scatterplot comparison of the performance of IPSec using TCP_RR Tests 72

Figure D.16 Boxplots comparison of IPSec using TCP_STREAM Tests 73

Figure D.17 Histogram comparison of IPSec using TCP_STREAM Tests 74

Figure D.18 Scatterplot comparison of IPSec using TCP_STREAM Tests 75

Figure D.19 Boxplots comparison of the performance of IPSec using UDP_STREAM Tests 76

Figure D.20 Histogram comparison of the performance of IPSec using UDP_STREAM Tests 77

Figure D.21 Scatterplot comparison of the performance of IPSec using UDP_STREAM Tests 78

Figure D.22 Boxplots comparison of the performance of IPSec using TCP_CC Tests 79

Figure D.23 Histogram comparison of the performance of IPSec using TCP_CC Tests 80

Figure D.24 Scatterplot comparison of the performance of IPSec using TCP_CC Tests 81

Figure D.25 Boxplots comparison of the performance of IPSec using TCP_CRR Tests 82

Figure D.26 Histogram comparison of the performance of IPSec using TCP_CRR Tests 83

Figure D.27 Scatterplot comparison of the performance of IPSec using TCP_CRR Tests 84

Figure D.28 Boxplots comparison of the performance of IPSec using TCP_RR Tests 85

Figure D.29 Histogram comparison of the performance of IPSec using TCP_RR Tests 86

Figure D.30 Scatterplot comparison of the performance of IPSec using TCP_RR Tests 87

Figure E.1 Boxplots comparison of MYSEA processes using TCP_STREAM Tests 90

Figure E.2 Histogram comparison of MYSEA processes using TCP_STREAM Tests 91

Figure E.3 Scatterplot comparison of MYSEA processes using TCP_STREAM Tests 92

Figure E.4 Boxplots comparison of the performance of IPSec using UDP_STREAM Tests 93

Figure E.5 Histogram comparison of the performance of IPSec using UDP_STREAM Tests 94
Figure E.6 Scatterplot comparison of the performance of IPSec using UDP_STREAM Tests 95
Figure E.7 Boxplots comparison of the performance of IPSec using TCP_CC Tests 96
Figure E.8 Histogram comparison of the performance of IPSec using TCP_CC Tests 97
Figure E.9 Scatterplot comparison of the performance of IPSec using TCP_CC Tests 98
Figure E.10 Boxplots comparison of the performance of IPSec using TCP_CRR Tests 99
Figure E.11 Histogram comparison of the performance of IPSec using TCP_CRR Tests 100
Figure E.12 Scatterplot comparison of the performance of IPSec using TCP_CRR Tests 101
Figure E.13 Boxplots comparison of the performance of IPSec using TCP_RR Tests 102
Figure E.14 Histogram comparison of the performance of IPSec using TCP_RR Tests 103
Figure E.15 Scatterplot comparison of the performance of IPSec using TCP_RR Tests 104
Figure E.16 Boxplots comparison of MYSEA processes using TCP_STREAM Tests 105
Figure E.17 Histogram comparison of MYSEA processes using TCP_STREAM Tests 106
Figure E.18 Scatterplot comparison of MYSEA processes using TCP_STREAM Tests 107
Figure E.19 Boxplots comparison of the performance of IPSec using UDP_STREAM Tests 108
Figure E.20 Histogram comparison of the performance of IPSec using UDP_STREAM Tests 109
Figure E.21 Scatterplot comparison of the performance of IPSec using UDP_STREAM Tests 110

Figure E.22 Boxplots comparison of the performance of IPSec using TCP_CC Tests 111

Figure E.23 Histogram comparison of the performance of IPSec using TCP_CC Tests 112

Figure E.24 Scatterplot comparison of the performance of IPSec using TCP_CC Tests 113

Figure E.25 Boxplots comparison of the performance of IPSec using TCP_CRR Tests 114

Figure E.26 Histogram comparison of the performance of IPSec using TCP_CRR Tests 115

Figure E.27 Scatterplot comparison of the performance of IPSec using TCP_CRR Tests 116

Figure E.28 Boxplots comparison of the performance of IPSec using TCP_RR Tests 117

Figure E.29 Histogram comparison of the performance of IPSec using TCP_RR Tests 118

Figure E.30 Scatterplot comparison of the performance of IPSec using TCP_RR Tests 119

Figure F.1 Boxplots comparison of APS and TPS on different servers using TCP_STREAM Tests 122

Figure F.2 Histograms of APS and TPS on different servers using TCP_STREAM Tests 123

Figure F.3 Scatterplots of APS and TPS on different servers using TCP_STREAM Tests 124

Figure F.4 Boxplots comparison of APS and TPS on different servers using UDP_STREAM Tests 125

Figure F.5 Histograms of APS and TPS on different servers using UDP_STREAM Tests 126

Figure F.6 Scatterplots comparison of APS and TPS on different servers using UDP_STREAM Tests 127

Figure F.7 Boxplots comparison of APS and TPS on different servers using TCP_CC Tests 128

Figure F.8 Histograms of APS and TPS on different servers using TCP_CC Tests 129

Figure F.9 Scatterplots comparison of APS and TPS on different servers using TCP_CC Tests 130

Figure F.10 Boxplots comparison of APS and TPS on different servers using TCP_CRR Tests 131

Figure F.11 Histograms of APS and TPS on different servers using TCP_CRR Tests 132

Figure F.12 Scatterplots comparison of APS and TPS on different servers using TCP_CRR Tests 133

Figure F.13 Boxplots comparison of APS and TPS on different servers using TCP_RR Tests 134

Figure F.14 Histograms of APS and TPS on different servers using TCP_RR Tests 135

Figure F.15 Scatterplots comparison of APS and TPS on different servers using TCP_RR Tests 136

Figure G.1 Boxplot showing the summary of test results for UDP_STREAM test . 139

List of Tables

Table 4.1 Configurations of TPE, server gateway and MYSEA server 24

Table 4.3 An explanation of the global options used in netperf 27

Table 4.4 TCP_STREAM test-specific options 28

Table 4.5 TCP_CRR test-specific options 28

Table 4.6 TCP_CC test-specific options 29

Table 4.7 TCP_RR test-specific options 30

Table 4.8 UDP_STREAM test-specific options 30

Table 5.1 Netperf "OMNI" test selectors used during benchmarking MYSEA . . . 33

Table 5.2 Summary of the average overhead witnessed during TCP benchmarks (Figure 5.1, 5.2, 5.2) 35

Table 5.3 Summary of IPSec results 36

Table 5.4 Summary of MYSEA results 37

Table G.1 Examples of UDP_STREAM Test raw data for NO IPSec encryption . . 137

Table G.2 Examples of UDP_STREAM Test raw data when IPSec encryption is used 138

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

WebDAV Web Distributed Authoring and Versioning
IPSec Internet Protocol Security
MYSEA Monterey Security Architecture
SSS-C Secure Session Server (Child)
SSS-P Secure Session Server (Parent)
COTS Commercial-off-the-shelf
GOTS Government-off-the-shelf
HTTP Hypertext Transfer Protocol
IMAP Internet Message Access Protocol
SLAT Single Level At a Time
SMTP Simple Mail Transfer Protocol
SPEC Standard Performance Evaluation Corporation
VoIP Voice over Internet Protocol
APS Application Protocol Service
BLP Bell and La-Padula
CIA confidentiality, integrity and availability
DAC Discretionary Access Control
DSS Dynamic Security Services
FSD Federated Services Daemon
I&A Identification and Authentication
IKE Internet Key Exchange
KPI Key Performance Indicator
LAN Local Area Network
MAC Mandatory Access Control
MLS Multilevel Security
NAT Network Address Translation
TCB Trusted Computing Base
TCP Transmission Control Protocol
TPE Trusted Path Extension
TPS Trusted Path Server
UDP User Datagram Protocol
TC Test Configuration

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgements

I would like to express my gratitude to my co-advisors; Professor Cynthia Irvine and Dr. Mark Gondree, for their patience and guidance through the course of this thesis. I would also like to thank my Singapore sponsor, the Defence Science and Technology Agency (DSTA), for giving me the opportunity to pursue my postgraduate studies, as well as the professors and lecturers at NPS who have taught me, and from whom I have learned a lot. I am also grateful to my family, especially my wife for giving me support and continuous encouragement during my course of study at NPS.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Demand exists for the mandatory enforcement of confidentiality and integrity policies, especially in the military and business domains. The need to share information across different sensitivity domains necessitates the development of multilevel distributed security architectures that enforce information flow policies, while maintaining usability and efficiency. Military and commercial systems are currently unable to provide high assurance support for multilevel security policy enforcement [1]. With adversaries and hackers performing increasingly sophisticated attacks, the lack of robust security measures are a cause for concern due to the risk of leakage of sensitive information, corruption of critical data and denial of service. New trusted computing approaches involving interoperable system security features and standardized security mechanisms are required to secure mission-critical systems.

MYSEA provides simultaneous access to different security classifications of information using standardized commercial-off-the-shelf components. Mandatory enforcement of confidentiality and integrity policies necessitates the design of distributed architectures to control information flow between systems while providing for security controls to protect against malicious code and attacks from adversaries.

1.1 Motivation

The confidentiality, integrity and availability (CIA) of data, applications and networks are vital to any organization. Mandatory access controls, where data is segregated into security classifications and is only accessible to suitably authorized personnel are a way of addressing security concerns associated with malicious software. The need to protect sensitive data and allow selective sharing of information between users necessitates the development of high assurance trusted systems. For users to embrace the security controls imposed, systems need to be efficient and user-friendly. Therefore, there is a need to investigate the system performance of MYSEA, to identify and analyze the potential bottlenecks.

1.2 Purpose of Study

Various trusted security components are part of MYSEA. These amend the underlying platform to provide authorized users with simultaneous access to information of different classification

levels, introducing necessary overhead to the system. There is a need to investigate the "cost of security" and study the performance overhead associated with high assurance systems following this design, compared with systems operating traditional servers. This is especially crucial for multi-user distributed systems where scaling and user-friendliness become important factors in user-acceptance and ergonomic security, both of which are stated MYSEA goals.

To create a trusted security environment, the MYSEA design introduces some (perhaps significant) overhead that may impact performance. Every TCP connection (for User Datagram Protocol (UDP), every packet) from the multilevel Local Area Network (LAN) is processed, verified and cross-referenced against access control permissions by a trusted subject. This inspection takes place in different processes in the server architecture. Key Performance Indicator (KPI)s associated with the system need to be measured and their impacts assessed. The "cost of security" in terms of efficiency, performance and responsiveness is an important factor in the application and risk analysis of security controls. Thus, measurement of the KPIs is needed before an informed tradeoff between the performance penalties associated with implementing MYSEA and the security requirements and policies mandated.

The goal of our study is to analyze the performance of selected aspects of MYSEA and, when applicable, identify system performance bottlenecks. System usability is determined by many factors, one of which is system *latency* measured by the end-to-end time taken for a system to react to a user's command. Another important factor is to gauge the *throughput* of the system, measuring the bytes per second processed by the system. Benchmarking the distributed MYSEA system will help to reveal potential design inefficiencies so that, in the long run, steps can be taken to improve its performance and efficiency. A bottleneck occurs when the performance of a system is limited by a number of components or resources, thus reducing the overall throughput or increasing the latency experienced by a user. In the absence of bottlenecks, our secure system performance study can be interpreted as characterizing the "cost of security" in a multilevel security context.

1.3 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 provides the background on MYSEA and an overview of its trusted components and services, which in combination form the multilevel secure distributed system. It also provides a detailed discussion of the design of MYSEA and the design considerations that impact its performance. Chapter 3 outlines the objectives of the thesis by identifying the high-level and detailed requirements for benchmarking

MYSEA, including the criteria for selection of a benchmark tool. These requirements provide the basis for the selection of the KPIs that were measured. This chapter also analyses the various benchmark tools for consideration; compares the pros and cons of each tool and selects a tool to benchmark MYSEA. It describes the benchmark methodology used to ensure that tests were conducted fairly, accurately and without bias. Chapter 4 describes the test environment, its test configurations and the test plan used for the execution of the benchmarks. Chapter 5 deals with the actual implementation of the benchmarks and analyzes the results collected by the various benchmark tests. Chapter 6 concludes by summarizing the results, discussing related work and providing recommendations for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Monterey Security Architecture Overview

MYSEA, The Monterey Security Architecture, is a distributed client-server architecture composed of existing Government-off-the-shelf (GOTS) and Commercial-off-the-shelf (COTS) software, open-source components, and (relatively few) special-purpose, high-assurance components [1] [2] [3] [4]. MYSEA provides a trusted computing operating environment for enforcing information-flow policies in a multilevel LAN. MYSEA allows vertical integration of application security requirements with underlying security services; an existing Quality of Security Service model [5] and framework are applied to the integrated security structure. In addition, MYSEA supports a secure and unforgeable bidirectional connection, called a *trusted path*, between the user and the trusted elements of the systems of the system, as well as trusted channels between devices.

MYSEA enforces confidentiality and integrity policies formalized in the Bell and La-Padula (BLP) and Biba security policy models respectively. The high assurance MYSEA Servers enforce the security policies and control the information that a user can access at a particular sensitivity level. These servers host various open-source or commercial application servers (i.e., web server and mail server), providing services along with multilevel views of information to clients. MYSEA currently supports Simple Mail Transfer Protocol (SMTP) [6], Internet Message Access Protocol (IMAP) [7], Voice over Internet Protocol (VoIP) [8], Web Distributed Authoring and Versioning (WebDAV) [9], Hypertext Transfer Protocol (HTTP) [10] and Wiki [11].

MYSEA clients are commodity PCs equipped with specialized Trusted Path Extension (TPE) devices. TPEs are inserted between unsecured client workstations and the MYSEA server. Every network connection from a MYSEA client to the Multilevel Security (MLS) servers is labeled, thus permitting the enforcement of the security policies for data entering MYSEA Server. Together, the MYSEA Servers and TPEs form the Trusted Computing Base (TCB).

Section 2.1 provides the MYSEA design overview and Section 2.2 describes a typical usage scenario of how clients gain access to the services hosted on MYSEA server; Section 2.3 provides a detailed description of the important components and services in MYSEA and possible sources of performance overhead. Section 2.4 outlines the security policy enforced in MYSEA and related design decisions. Section 2.5 describes the current MYSEA prototype.

2.1 MYSEA Design Overview

MYSEA is designed to provide application services in a high assurance, distributed environment, while protecting users from malicious code and other attacks. The design of MYSEA allows mandatory policy enforcement mechanisms to be allocated to a few specialized high-assurance trusted components, while encompassing other low assurance commercial components and applications to support operational functionality.

The key design principles of MYSEA are (1) security dependencies must be partially ordered; (2) the ordering of trust and trustworthiness must be present and (3) the principle of least privilege must be enforced. A federation of highly trustworthy MLS servers provides the locus of policy enforcement, mediating all accesses to resources in the system. The MYSEA servers host various open-source or commercial services such as mail and web services. The MYSEA federation is designed to be a robust and scalable foundation supporting expansion: the scalability requirement for MYSEA is to support up to 100 clients with concurrent user sessions at 100 different security levels.

To establish a secure communication channel between clients and MYSEA servers, trusted TPEs authenticate and disambiguate clients. All network traffic flows in and out of MYSEA are tracked to determine the security level of the client making the request. This results in an implicit labeling of all network traffic.

MYSEA servers and TPEs collectively enforce the mandatory security policies, forming a Trusted Computing Base (TCB). The MYSEA TCB is designed to facilitate evaluation by TCB subsets [12]. TCB subsets allow for the partition of a complex system into smaller disjoint trusted subsets, each of which resides in its own protection domain and enforces a security policy subset upon the subjects and objects under its control [12]. The composition of each TCB subset is small and less complex, and can be evaluated individually for policy enforcement correctness. This chain of incremental evaluations of TCB subsets combined with domain mechanism, which in the case of MYSEA is both logical and physical, ensures that the policies are enforced.

2.2 MYSEA Concept of Operations

Users logon to the MYSEA server using a Single Level At a Time (SLAT) client workstation. The commodity workstations can be on the local multilevel LAN as well as on legacy single level networks. Clients on the multilevel LAN must have an associated TPE. Labels for clients

on legacy networks are implicit, as all clients are at the level of the network itself. Each LAN-based client workstation and TPE is logically and physically bound together. There is exactly one TPE for each client workstation, and no more than one user at a time can be active on the given TPE. The TPE distinguishes user sessions and establishes a trusted path of communication to the MYSEA server. Upon logon, the TPE establishes an identity for audit and access control purposes; and forwards the user's credentials to MYSEA server.

Next, the user uses the TPE to negotiate a session level within his security clearance range. Based on the security policy enforced by the MYSEA server, the session level determines the domain and resources he can access during that session. For example, to access resources at a higher classification level, the user would need to change his session level using the TPE.

Following session level negotiation, the SLAT client-TPE pair can be used to access applications hosted on any MYSEA server or on a single-level network at the user's session level. When activity at a particular session level is over, data created or modified on the client is stored on the MYSEA server, and the SLAT client state is automatically purged at the end of each session.

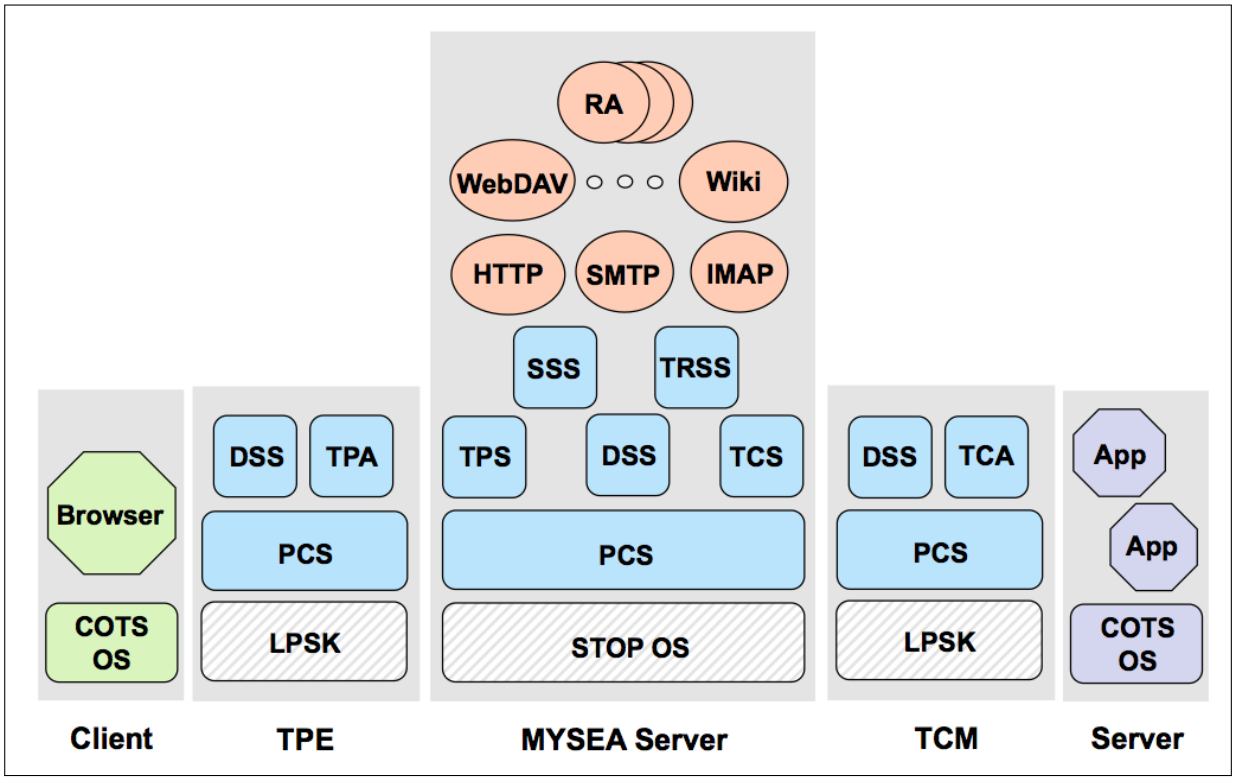

Figure 2.1: Major Components in MYSEA From [1]

2.3 Overview of MYSEA Components and Services

This section describes the major MYSEA components and services that comprise MYSEA's design. Figure 2.1 shows the relationship between some major components and services in MYSEA. The major components in MYSEA include:

1. **Clients and TPEs.** Each SLAT client executes popular software applications and productivity tools such as word processors and web browsers. The TPEs controls the interface between the client and the MYSEA server. The TPE provides transport-layer security, trustworthy identification and authentication, and session establishment.
2. **MYSEA Trusted Proxy Service.** Various untrusted, open source applications hosted on the MYSEA server provide services (e.g. web services and mail services) to clients. The clients cannot directly access these applications. Instead, clients invoke the services through a layer of proxy processes provided by MYSEA.
3. **MYSEA server Federation.** The MYSEA architecture is distributed, leveraging a cluster of high-assurance MYSEA servers. These servers provide the locus of multilevel security policy enforcement and host various open source or commercial application protocol servers.
4. **Dynamic Security Services.** The Dynamic Security Services (DSS) manages the transport-layer security used to protect the trusted path between the TPE and remote MYSEA server.
5. **Single-Level Networks.** Existing classified single level networks are connected to the MYSEA server providing application services to both local clients and clients on legacy networks.

2.3.1 Clients and TPES.

MYSEA clients are SLAT, and can only access and process data, at a single session level at one time. MYSEA clients consist of two components: commodity PCs and TPEs. Commodity PCs are untrusted, "thin-client" machines, running familiar commercial software and OSes. They are loaded with COTS products and open-source software that assist the user to perform tasks like word-processing, web browsing and communication. With this software, users can access remote applications hosted on a MYSEA server or single-level network.

The TPEs operate largely under the direction of the remote MYSEA server and help to enforce MLS LAN policies. All network traffic in and out of a SLAT Client is routed through its TPE. The TPE ensures that a secure tunnel is established between client and TCB. This tunnel is

encrypted with Internet Protocol Security (IPSec) leveraging DSS services (see Section 2.3.4).

When a remote session is established, the TPE sends a request for a service via the secure tunnel to the MYSEA server. The TPE acts as a Network Address Translation (NAT) device, performing address translation on all traffic traversing between the client and the MYSEA server. On session establishment, the TPE passes the user's credentials to the MYSEA server, which validates the credentials and instructs the TPE whether to allow or deny access. The user may use the TPE to send commands to the MYSEA server to perform session-related tasks such as ending the session or changing session level.

2.3.2 MYSEA Trusted Proxy Service

Each MYSEA server consists of a high assurance operating system that enforces the mandatory security policy. The high assurance kernel creates labeled protected domains and associates security attributes with active and passive entities exported at its interface [4]. This results in a distinct separation of data into different security classification domains. In MYSEA, trusted multilevel processes control the invocation of single-level applications that interact with the user at his current session level. Various trusted processes are created by a daemon during system initialization and are responsible for the proxying and mediating network access on behalf of the application service, allowing it to communicate with the remote client.

For every application service hosted on MYSEA, a Secure Session Server (Parent) (SSS-P) is created to respond to connection requests from clients. The SSS-P monitors the network and is responsible for accepting and verifying that service requests from a TPE have a valid session. The SSS-P spawns a proxy service (Secure Session Server (Child) (SSS-C)) for *each* client connection request. This proxy sets up the actual untrusted Application Protocol Service (APS) associated with the request, e.g. an untrusted Apache web service is an APS at the level of the remote user's session. All subsequent communication between the client and the APS is sent and received via the proxy SSS-C, using one or more "MYSEA sockets".

2.3.3 MYSEA Server Federation

The MYSEA server federation is a cluster of high assurance servers that provide the locus of multilevel security policy enforcement and hosts various open source or commercial services. During session establishment, the TPE forwards the user's credentials to the MYSEA server hosting the Trusted Path Server (TPS) process. The TPS maintains session data including the user credentials, user's clearance range and his current session level. The TPS process is respon-

sible for validating connection requests from the TPE, and handles TPE requests for updating session information (like login, change session level, run and logout), ensuring that accountability aspects of the security policy are enforced.

In the MYSEA server federation, a client can request an APS service hosted on a server that is not hosting the TPS process. A Federated Services Daemon (FSD) on each MYSEA server handles the internal communication required, validating the authenticity of the client and mediating the accesses to the requested resources according to the security policies.

2.3.4 Dynamic Security Services

DSS is an IPSec configuration and administrative tool that assists in the creation of secure tunnels between TPEs and the MYSEA server. Each DSS client resides on the TPE and connects to the DSS server on the MYSEA server. The DSS admin tool allows a security administrator to log onto the MYSEA server, set and manage the security policies associated with the IPSec tunnels.

2.3.5 Single-level Networks

Users on a legacy network can only access applications on the MYSEA server at the classification level associated with the label of the corresponding network device. Similarly, MYSEA clients can also access applications on the legacy networks. This allows existing applications on legacy networks to run unmodified and be accessible to SLAT client on the MLS LAN, with the MYSEA TCB mediating all accesses.

2.4 MYSEA Security Policies

MYSEA supports both Mandatory Access Control (MAC) and Discretionary Access Control (DAC) [1]. The system MAC policy is a confidentiality and integrity information flow control policy, where the sensitivity labels form a lattice [13]. The confidentiality and integrity policies are formally modeled by the BLP [14] and Biba [15] models, respectively. The high-assurance OS of the MYSEA server mediates access to files and other labeled system resources. Network connections on the MLS LAN are mediated by small, trusted processes on the MYSEA Server, using information about client session level. For example, a client at SECRET cannot communicate with a client at UNCLASSIFIED, nor can a client at SECRET communicate using a port in use by an untrusted APS acting on behalf of a remote client at UNCLASSIFIED. Thus, access to network resources is impacted by checks performed by MYSEA processes, and access to system resources is impacted by checks performed by the underlying high-assurance OS.

For Discretionary Access Control (DAC), the MYSEA server allows users' processes to modify permissions associated with data objects such as files via runtime functions. The permissions are registered with the server, and access decisions are made based on the end-users' identities bounded to the processes. This policy is less critical than MAC; hence its enforcement can be allocated to the applications hosted on MYSEA server. Users can determine the access controls of APS processes to access resources such as files. The access decisions are based on the MYSEA user identities associated with the processes and permissions that are registered with MYSEA server. This policy can be modified during runtime.

2.5 Current MYSEA Prototype

To benchmark MYSEA, we will analyze the existing MYSEA prototype system. Here, we highlight some consequential differences between the current prototype system, and the MYSEA design. Our benchmark tests will be performed on the current prototype.

2.5.1 Clients and TPEs

There are several designs capable of satisfying the MYSEA client component. The TPE application and SLAT client could both run on a trusted separation kernel or hypervisor: each single-level client runs in its own partition; the TPE application runs in its own partition. Alternatively, these components could be implemented using a small, handheld tactical device that connects the stateless, thin client to the MLS LAN. The current prototype system follows this latter strategy: the TPE mediates access to the network for a stateless client (currently a commodity PC that must be manually purged of state whenever a session ends). The prototype TPE application is hosted on Linux, running on a hand-held device.

2.5.2 MYSEA Server Federation

Each MYSEA server is designed to run on an existing, commercial high-assurance system. Previous MYSEA prototypes utilized BAE XTS-400 running STOP 6, an operating system with CC EAL5+ certification [16]. The current prototype uses the next generation of this product: STOP 7, which attained a CC EAL4+ certification [17].

2.5.3 Dynamic Security Services

The security requirements of MYSEA necessitate the establishment of a secure tunnel for network communication between the TPE and MYSEA server. As STOP 7 lacks IPsec support, another mechanism for establishing a secure tunnel between the client and server is needed. A host-to-gateway IPSec tunnel is established between the TPE and a Server Gateway device.

All network traffic to the MYSEA server is routed through the Server Gateway. Together, the MYSEA server and Server Gateway form a single logical unit. The DSS client on both the TPE and Server Gateway control this IPsec tunnel, modulating the tunnel's characteristics based on requests from a remote DSS admin tool. The DSS Client uses Racoon [18], the user-space IPSec daemon, to provide the dynamic protection services.

CHAPTER 3:
Objectives and High-Level Analysis

This chapter outlines the factors that are considered during the conceptual and planning phases for benchmarking MYSEA as well as details about the selection of the tool used for performance measurement.

We are interested in the overhead introduced by MYSEA's design, such as those introduced by processes performing Mandatory Access Control (MAC) checks associated with network related calls. We attempt to measure components individually to determine which act as system bottlenecks. By isolating and measuring the behavior of individual components, rather than measuring the overall system or application performance, we are able to quantitatively assess MYSEA's design. Figure 3.1 outlines the process flow resulting from a client's requests to servers in MYSEA.

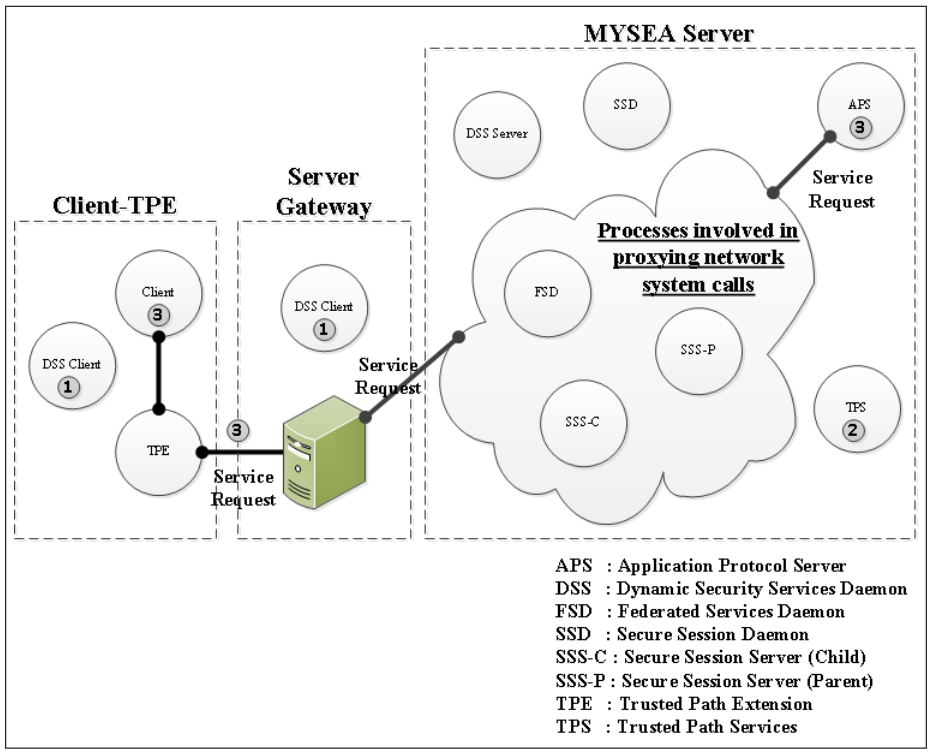

Figure 3.1: Process flow of a client making a service request to MYSEA

We describe the events that ensue next.

1. **Establish IPSec Tunnel**

   The DSS client on the TPE establishes an IPSec tunnel with the Server Gateway for the MYSEA server, forming a trusted communications path between the client and MYSEA. The DSS client on the TPE operates as a slave, following the DSS policy dictated by the MYSEA server.

   We can measure the overhead introduced as every network packet is inspected and analyzed according to the loaded IPSec policy. If the communication is allowable by the security policy, every network packet is encrypted at layer three of the TCP/IP protocol stack and forwarded to the MYSEA Server via the secure communications channel.

2. **Login and Start Secure Session**

   Once an IPSec tunnel has been established, the user logs on to MYSEA server using the TPE and establishes a session at some security level. The TPS process on MYSEA is responsible for identification and authentication (I&A) and must monitor for TPE connection requests. There is a TPS Parent process for the MYSEA federation, which manages all Identification and Authentication (I&A) and login requests. The TPS Parent accepts login requests from MYSEA clients and spawns a TPS Child to handle trusted path communications for each client. When the TPS child receives data from the client, it reads and processes the client's command and returns the output before waiting to receive the next command. Upon the successful I&A, the user at the MYSEA client negotiates a session within his security range to access the services hosted on the MYSEA server.

   The user can issue a "Run" or "Logout" command via the TPE. The latter terminates the secure session and the TPS on MYSEA takes actions to update its information about active sessions. Various interactions can take place during the login process, resulting in overhead impacting the performance of MYSEA.

3. **Service Request**

   Every service request made from the client to the APS goes through a series of proxy calls before it is processed by the APS. The APS is the application process that responds to the service request, and cannot directly communicate with the client. The APS is an untrusted process; the SSS-C process mediates all network connections. Communications between the APS and SSS-C are handled through a MYSEA socket (MSKT) allocated by the TPS Child process. When the APS requires a socket call, the APS updates the MSKT database, and signals the SSS-C. The SSS-C retrieves the request from the database, responds to the client's service request and inserts any retrieved data into the database. The APS is then notified to retrieve the requested data from the database. Each TCP connection (for UDP,

each packet) is checked using its source and destination, to see (in the context of active sessions and outstanding network use) if the information flow is allowable according to the MAC policy. When the APS and TPS processes are deployed on different physical servers, there may be additional overhead due to the communication between MYSEA servers, querying about the status of active sessions within the Federation. These proxy design decisions may impact the performance of MYSEA.

We are interested in measuring the performance impact of the following three components: **(1) IPSec**, **(2) the MYSEA trusted proxy** and **(3) MYSEA's federation of servers** as a result of the events described earlier. IPSec is related to the cost of IPSec encryption of all network traffic between the TPE and server gateway. Measuring the MYSEA trusted proxy allows us to analyze the cost of the client making a service request to MYSEA. We can measure the overhead associated with the proxying of system calls for making any service request. Targeting the MYSEA federation allows us to measure the cost of maintaining state among the servers.

The output of a benchmark must be reproducible, consistent, unbiased and realistic, based on sound scientific methods and principles. The demands for a successful benchmark imply a need for proper scientific procedure and a need for common agreement on the metrics [19]. Therefore, the benchmarking methodology adopted in this thesis consists of a four stage process: (1) determine the performance indicators; (2) select or develop tools for the benchmark; (3) configure the test environment and; (4) execute benchmarks and analyze output. We describe stages (1) and (2) in this chapter, stage (3) in Chapter 4 and stage (4) in Chapter 5.

3.1 Benchmarking Methodology

Benchmarking is the process of comparing business processes, services or products against one another. It allows organizations to understand quantitatively the performance of the targets, possibly for the purpose of improving their characteristics. Benchmarking of an IT system is the process of running a specific program or workload on a specific machine or system and measuring the resulting performance [20].

There are a wide variety of approaches to benchmarking because different approaches are necessary to evaluate individual aspects of a system. For example, synthetic benchmarks load or stress test some specific targeted components in the system; application benchmarks measure performance of the system as it accomplishes some real-world task (like running a web server) under some synthetic load. Benchmarks can be divided into the following categories [19] [21] [22]:

- **Application benchmarks**, also known as macro benchmarks, take an application and measure the time required to complete some tasks. They measure end-to-end performance on a specific workload, assessing the performance experienced by end-users. For example, the library generator ATLAS [23] uses application benchmarking to build linear algebra libraries automatically tuned for the target processor. An evaluation of the Denali Isolation kernel [24] made use of web server benchmarks to compare that platform with BSD. The SPEC CPU benchmark suite measures several applications and combines the results to rank overall performance.
- **Micro-benchmarks** measure the performance of individual operations in isolation. Benchmarks intended to measure the performance of kernel primitives; system calls and the network stack are commonly measured using micro-benchmarks. Ousterhout [25] developed benchmarks to isolate a number of kernel primitives providing measurements of performance, independent of the machine's platform. Micro-benchmarks were also used to quantify the performance of the Denali Isolation Kernel's primitive operations [24].
  Network micro benchmarks measure the bandwidth, throughput and network latency experienced by the system, irrespective of application. Such benchmarks provide insight into low-level network stack performance; in particular the latency associated with opening a socket connection. A performance characterization of the Chelsio T110 10-Gigabit Ethernet [26] adaptor makes use of micro-benchmarks to measure the single and multiple connections of point-to-point latency and unidirectional throughput.
- **Other benchmarks** target the hardware used by the system, or the end-users of the system. Benchmarking I/O performance provides a measurement of the cost of retrieving or storing data using some particular hardware. It may measure sequential and streaming I/O bandwidth for single processes, or random I/O latency for multiple processes (including the time taken to create, write and read a number of files in a file system).
  Subjecting real-world users to the system provides realistic performance measures. These user tests produce results that are not always comparable across systems or versions, as these tests cannot be easily repeated.

3.2 Performance Indicators

Different tools measure different performance indicators such as processor speed, memory latency, network throughput, bandwidth and socket connection latency. Chapter 2 describes the MYSEA process structure and their relationships. For a user to access an APS service hosted on MYSEA, secure communication is established between the MYSEA client and MYSEA server

using IPSec encryption. Every network packet is inspected and encrypted at the TPE before it is transmitted to the server gateway. At the server gateway, network packets are decrypted and forwarded to the MYSEA server for processing. Also, during the initial setup of APS communication, the SSS-P listens on the network for requests and spawns a proxy to handle the connection. The proxy acts as an intermediary between the APS and network to handle the request. All traffic is inspected and proxied within MYSEA before the APS can receive it. In addition, the TPS process that handles all I&A can be deployed on a different server from those hosting MYSEA's APS services. If a service request is made to an APS hosted on a server other than the TPS, internal communication between the servers is required to assert the identity and authenticity of the request. The following two KPIs are identified to measure the performance of MYSEA. First, we can measure the TCP and UDP throughput transfer rates of the IPSec encryption, MYSEA proxy and MYSEA federation. Second, we can also measure network latency, the time taken for data to be transmitted from one point to another. In MYSEA, one could measure the latency associated with opening and closing a network socket, reflecting the initial costs of the setup for a connection between the client and server. We can also measure the latency associated with the time taken for a request and response transaction to complete.

3.3 Benchmarking Tools for Use in MYSEA

There are various benchmarking tools available to measure the performance of systems that provide networked services such as MYSEA. Examples include Apache Jmeter, LMbench and Netperf. These benchmark tools automate the process of running performance tests for the targeted system. These tools are intended to produce repeatable results, allowing for consistent analysis of data to compare system performance.

Any benchmark tools used in the context of MYSEA must satisfy certain operational requirements. The current MYSEA prototype runs on the STOP operating system, a proprietary high assurance operating system that is binary compatible with Linux. Any application service hosted on MYSEA needs to be modified to run in a specific inetd mode, integrated with MYSEA processes. Therefore, source code for the tool selected must be available, and compatible in the MYSEA environment.

The tool selected must be able to measure specific targets of the MYSEA system. We desire to characterize each MYSEA component to analyze each feature of MYSEA's design. In particular, an end-to-end application benchmark tool will not be able to produce such measurements, as it characterizes the system only in terms of its performance with a specific application. For

example, MYSEA currently supports a variety of application services (web services, VoIP, etc.), and the benchmark tool selected must not characterize the performance of the application service. Additionally, as MYSEA is an experimental system without actual users, we are unable to develop workloads representative of its use in a realistic operating environment. The following sections described the benchmark tools considered.

3.3.1 Jmeter

Apache Jmeter (TM) [27] is an open-source web server benchmark written in Java designed to test functional behavior and measure performance. It can be used to test performance on static and dynamic web resources, and perform load testing on the server. It is extensible and allows the development of plugins that provides extensible testing capabilities.

Jmeter can be configured to benchmark the performance of the existing web services on MYSEA at the client. This would eliminate the need to modify or adapt the tool to run on MYSEA. However, Jmeter is unable to produce results that measure isolated components of MYSEA, for example the cost of proxying specific network calls. This tool would characterize the performance costs of MYSEA's TCB, combined with the performance characteristics of the untrusted, third party web service, rather than characterizing the MYSEA system performance in isolation.

3.3.2 Netperf

Netperf [28] is an open-source benchmark tool written in C that is designed to characterize general network performance experienced in client-server settings. The tool is split into two components: netperf and netserver. The netperf tool is a client that requests tests to be performed, while the netserver tool is a daemon process that resides on the system under test and responds to requests. Netserver can run as either as a daemon, or as an inetd service. Netperf provides tests for both unidirectional and bidirectional throughput and end-to-end latency. It can be used to measure various aspects of networking performance with primary focus on unidirectional data transfer and request/response performance using TCP or UDP socket interfaces.

3.3.3 LMbench

LMbench [22] [29] is a suite of micro-benchmarks written in C designed to measure latency and bandwidth performance of system operations like data movement among the processor and memory, network, file system and disk. LMbench was developed to identify and evaluate system performance bottlenecks, and is portable across a wide set of Unix systems. LMbench is open-source and binary compatible with Linux. It can also be used to measure specific targeted

components and servers. However, LMbench lacks comprehensive documentation: its manual pages lack detail, and the documentation does not describe how each specific test measures its target. The lack of documentation hinders the porting of LMbench source code to work in the MYSEA environment.

3.3.4 SPEC

Standard Performance Evaluation Corporation (SPEC) is a non-profit corporation that develops and maintains a standardized set of relevant benchmarks for measuring the performance of IT systems. Organizations can measure the performance of their products or services using the SPEC benchmark suites. The results can be submitted to SPEC for validation, creating an independently validated benchmark. This provides a fair comparison between similar products and services. SPEC suites of benchmarks are written in C, Java or Fortran. However the source code cannot be modified under its licensing rules to prevent the optimization of benchmarks to skew the performance results positively, affecting fair comparison. This limits the portability requirement of the benchmark tool.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Methodology

This chapter describes the test environment, configurations and test plan used to benchmark selected key performance indicators in MYSEA. The tests are designed to produce results from which unbiased comparisons can be drawn. The benchmarks have been documented so that they can be repeated with the same experimental setup.

The test environment consists of the physical setup, which includes the servers, network routers, switches, and their interconnection. The software configuration used in the benchmark tests consists of the server operating systems, client operating systems and the MYSEA components and services.

We can configure different setups that allow for the measurement of targeted components (see Chapter 3) of the MYSEA server. The setups are formed by combining permutations of the three components, resulting in a total of eight different setups. We compare the eight different setups against one another. The comparisons of different setups are referred to as Test Configuration (TC)s where each TC measures the performance of the components and services in isolation.

We can execute a suite of benchmark tests on each TC, measuring the different aspects of performance of the isolated components and services. Netperf client executes the following five tests: **(1) TCP_STREAM, (2) TCP_CRR, (3) TCP_CC, (4) TCP_RR** and **(5) UDP_STREAM**.

We describe the tool selected and modifications that enable the benchmarking of MYSEA in Section 4.1. We describe the test environment for the benchmarks in Section 4.2, the various TCs in Section 4.3 and the suite of benchmark tests executed on MYSEA server in Section 4.4. Section 4.5 describes the test plan for performing the benchmarks.

4.1 Tools and Instrumentation

Netperf version 2.5.0 is selected as the tool used to benchmark MYSEA. There are three reasons behind the selection of netperf as the benchmark tool. First, netperf is open-source and, second, it is written in the C programming language. The necessary modifications to the netperf source code can be made to enable it to run on the current MYSEA prototype. Third, netperf has comprehensive documentation that describes in detail how each test performs the benchmark and what the test is measuring.

To benchmark MYSEA, we deploy the netperf client on the TPE, and netserver on the MYSEA server. The netperf client runs a suite of benchmark tests, making requests to the netserver daemon. Netserver responds to these requests, measuring latency and throughput performance and returns the results to netperf.

Netserver has two modes of operations: *inetd* and *daemon*. The *inetd* mode is used in TCs using MYSEA, since all MYSEA processes are started by a trusted *inetd* process. The daemon mode is used in TCs where MYSEA is not being measured, and it is not natural to start the process via inetd.

In the *inetd* mode of operation, the MYSEA trusted inetd process SSS-P spawns the netserver process when a request is received on the netserver service's listening port. Netserver has been modified to act as an untrusted MYSEA service and thus uses the MYSEA proxy service. This allows us to measure the proxy service performance. Refer to Appendix A for more details on the modifications to the netserver source code.

In the *daemon* mode of operation, one starts the netserver process manually before testing. The netserver daemon process monitors the network, responding to requests from the netperf client. This mode of operation does not require any modification to the netserver source code.

Appendix B provides details on how the netperf client and netserver server process are installed, and how the benchmark tests are executed.

4.2 Test Environment

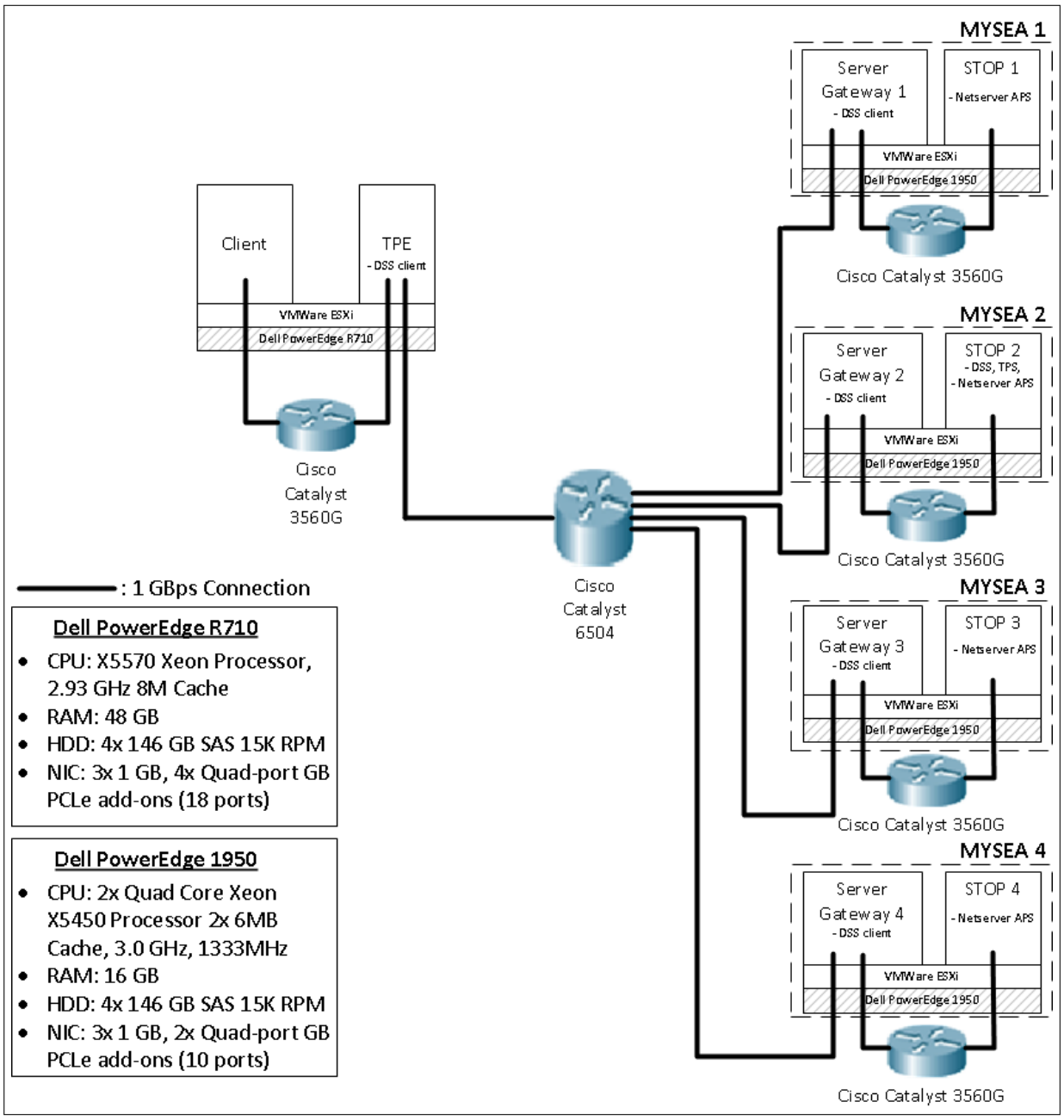

Figure 4.1: The MYSEA benchmark test environment (network configuration and hardware).

Figure 4.1 shows the configuration and topology of the multilevel LAN used during benchmarking. All network connections are one gigabit per sec (1Gbps) to reduce the possibility of

Table 4.1: Configurations of TPE, server gateway and MYSEA server

Server Name	Configuration
TPE	512MB RAM, VMWare ESXi v4.1.0 Fedora 7 linux kernel 2.6.23, IPSec Tools v0.6.6
Server Gateway	3GB RAM, VMWare ESXi v4.1.0 Fedora, IPSec Tools v0.6.6
MYSEA server	2GB RAM, VMWare ESXi v4.1.0 STOP v7.3.1

limited network bandwidth affecting the results of our benchmarks. The configurations of the TPE, server gateway and MYSEA servers are provided in Table 4.1.

4.3 Test Configuration

Different TCs are designed to measure and compare the performance the MYSEA components and services in isolation. Figure 4.2 compares all possible test configurations. The setups are formed by the combination of three pairs of scenarios: “I”/“I-”, “M”/“M-” and “F”/“F-”. Each scenario is described next.

“I” represents the scenario when IPSec is used to encrypt all network traffic between the TPE and the MYSEA server gateway. “I-” represents the scenario when no network traffic is encrypted in the IPSec tunnel. This is achieved by flushing all IPSec rules from Racoon. There are subtle differences between setups A5, B6, C7 and D8, although the TCs indicate that they measure the performance of IPSec encryption. A5 and B6 measure the performance of IPSec encryption with netserver spawned in inetd mode in a MYSEA environment where netserver is accessed via MYSEA trusted proxy calls. C7 or D8 is deployed as a daemon on STOP server.

“M” represents the scenario when the netperf request undergoes a series of proxy calls in MYSEA before reaching the netserver process, with netserver process spawned by the MYSEA trusted inetd process. A secure session is first established between the TPE and the MYSEA server before netperf client executes the suite of benchmark tests. “M-” represents the scenario when the netserver process is running as a daemon on STOP with no MYSEA processes running.

“F” represents the scenario when the TPS process and the netserver process are deployed on different physical servers in the MYSEA federation. This results in additional internal network communication between the servers when netperf client executes the tests. “F-” represents the scenario when the TPS process and our benchmark tool are deployed on the same physical server.

For example, “A5” compares the two test configurations “I M F” and “I-M F”. This test configuration measures the overhead associated with IPSec (“I” vs “I-”), in the configuration where MYSEA services are being used (“M”) on servers that require internal, federated requests (“F”).

Figure 4.2: Figure showing the comparisons of different test configurations. Test cases considered in this thesis are underlined

4.4 Types of Tests

Netperf is capable of running different types of tests to measure different aspects of performance of the targeted system. Five different tests are executed on the setups specified in Figure 4.2: TCP_STREAM, TCP_CRR, TCP_CC and TCP_RR, UDP_STREAM. These five tests form the suite of benchmark tests. [30] provides additional descriptions of the various tests. Each of these will be discussed in their respective sections below.

We made use of the "OMNI" test in netperf to run the various tests. Previous netperf versions included different source code to perform different tests. The "OMNI" test in netperf is a consolidation of all available test functions into a single test. Different flags allow different options to be selected to enable the execution of specific test functions. There are two types of flags: global and test-specific. "Global" flags (see Table 4.3) provide options that apply to all benchmark tests. Test-specific flags are described in their respective subsection.

Table 4.3: An explanation of the global options used in netperf

Flags	Value	Description
-H	*hostname*	This option sets the hostname or IP address of the targeted system, where netserver daemon process is deployed. This option allows us to measure the performance of the two different physical servers under the scenarios: "F" and "F-".
-P	0	A value of "0" for this flag disables the display of the test banner. The test banner was disabled in all of our experiments because we executed the benchmark tests multiple times in succession and the test banners are redundant and unnecessarily cluttered the output
-p	*port*	This option sets the port number for netperf client to use to connect to the netserver daemon process.
-o	*filename*	This option allows the netperf client to read in a file containing comma-separated header values. These headers correspond to the different output values we want to record in our results. Examples of these headers include "THROUGHPUT", "THROUGHPUT_UNITS", "PROTOCOL" and "ELAPSED_TIME".

4.4.1 TCP_STREAM

MYSEA clients access various services (i.e., web and mail) hosted on the MYSEA server using Transmission Control Protocol (TCP)-based protocols. The performance of MYSEA in handling TCP packets is important to the experience of the client, in terms of the latency and

throughput observed by the client. The TCP_STREAM test is designed to measure the performance of the targeted component as it processes TCP data packets. The test provides the throughput in bits per second and accounts for both the time required to push data across the network and the time to process it on the targeted component. This test is unidirectional and the time spent establishing the connection is not included in the throughput calculation. Table 4.4 shows the test-specific options used in netperf for the test: TCP_STREAM.

Table 4.4: TCP_STREAM test-specific options

Flags	Value	Description
-d	send	This option sets the direction of the test relative to the netperf process. A value of "send" causes netperf to send data to the netserver daemon process, resulting in a unidirectional test.
-T	TCP	This option sets "TCP" to be the protocol used for the test.

4.4.2 TCP_CRR

The TCP_CRR test combines TCP_CC and TCP_RR and measures the performance of the targeted component in handling TCP connections. This test allows us to measure the performance overhead in transactions per second of the targeted component in completing the entire TCP process: establishing a connection via the three-way handshake, transferring one byte of request, receiving one byte of response and tearing-down the connection. We can then obtain the latency of one complete transaction in seconds by inverting the results. This test is designed to measure the end-to-end latency experienced by a client when making a service request. Table 4.5 shows the test-specific options used in netperf for the test: TCP_CRR.

Table 4.5: TCP_CRR test-specific options

Flags	Value	Description
-c		This option explicitly instructs netperf to include connection establishment and teardown in its measurement.
-d	send | recv	This option sets the direction of the test relative to the netperf process. A value of "send" causes netperf to send data to netserver daemon process, while the value of "recv" causes netserver daemon process to send data back to netperf. This results in a bi-directional receive and response test.
-T	TCP	This option sets "TCP" to be the protocol used for the test.

4.4.3 TCP_CC

The TCP_CC test isolates the speed at which connections can be opened and closed between the target and client using the TCP protocol. A three-way handshake establishes a reliable connection between the sender and receiver. The test is performed in a synchronous manner, with no data transferred between the two entities. This test continuously opens and closes connections between netperf and netserver, tracking the number of connections created. This test provides the latency in transactions per second, a measurement of the performance of the targeted component in establishing and tearing-down TCP connections.

The MYSEA trusted proxy introduces additional overhead for every new client connection. For each new client connection, a proxy SSS-C process is spawned to handle the request. This test measures the impact of spawning the additional process to handle the client request. Table 4.6 shows the test-specific options used in netperf for the test: TCP_CC. This test measures the latency associated with the opening and closing of a transaction.

Table 4.6: TCP_CC test-specific options

Flags	Value	Description
-c		This option explicitly instructs netperf to include connection establishment and teardown in its measurement.
-T	TCP	This option sets "TCP" to be the protocol used for the test.

4.4.4 TCP_RR

The TCP_RR test is a request and response test using the TCP protocol, where each request and response packet is one byte in size. It is bi-directional and measures the number of complete transactions per unit time that can be exchanged between two entities. This test is independent of the time taken to establish the initial TCP socket connection and measures the latency associated with the transfer of a byte of request and response request between the client and server. Table 4.7 shows the test-specific options used in netperf for the test: TCP_RR. This test is designed to measure the speed for uploading/downloading large files to MYSEA server (e.g., the ftp server), where the initial cost of setting up a connection is less significant.

4.4.5 UDP_STREAM

UDP is used as the underlying transport protocol for VoIP, and some other services provided by MYSEA. The UDP_STREAM test is similar to the TCP_STREAM test except that UDP

Table 4.7: TCP_RR test-specific options

Flags	Value	Description
-d	send | recv	This option sets the direction of the test relative to the netperf process. A value of "send" causes netperf to send data to netserver daemon process, whereas the value of "recv" causes netserver daemon process to send data back to netperf. Used in combination, this results in a bi-directional receive and response test.
-T	TCP	This option sets "TCP" to be the protocol used for the test.

is used as the mode of transport rather than TCP. This test provides the throughput in bits per second and measures the performance of the targeted components as they process UDP data packets. Table 4.8 shows the test-specific options used in netperf for the test: UDP_STREAM.

Table 4.8: UDP_STREAM test-specific options

Flags	Value	Description
-d	send	This option sets the direction of the test relative to the netperf process. A value of "send" causes netperf to send data to netserver daemon process, resulting in a unidirectional test.
-T	UDP	This option sets "UDP" to be the protocol used for the test.

4.5 Test Plan

Netperf client runs the suite of benchmark tests on the following six setups: **"I M-F", "I M-F-", "I-M-F", "I-M-F-", "I-M F"** and **"I-M F-"** (See Figure 4.2). Each benchmark test was executed sequentially for ten seconds (netperf default), and each test is repeated for one hundred trial tests. We conducted experiments in the initial laboratory setup and repeated the trials with different numbers of times: 10, 50, 100, 1000, 10000. One hundred (100) trials was selected as it optimizes the amount of time required for the completion of the entire suite of benchmark tests while achieving significant population samples with low variances. Before any execution of benchmark tests, the test environment was restarted and prepared for the respective setup. Steps included: recompilation of the netserver source code for the respective scenario ("I-M F", "I-M F-", "I-M F", "I-M F-", "I M-F" and "I M-F-"), configuration of IPSec to setup/teardown the secure tunnel, and deployment of netserver on the appropriate server. The results of the tests were collected at the end of experiment.

For the TCP_CC, TCP_CRR and TCP_RR benchmark tests, a two-minute delay wait period between successive tests was introduced to address the following problem. During initial trial experiments, abnormal results were obtained when these tests were executed. The latency of certain trials was several times lower than the rest of the trials. After several experiments, it was determined to be an issue associated with a TIME_WAIT reuse in the establishment of a socket connection. The latency drops dramatically when the test reuses a connection that is in the TIME_WAIT state. This results in a non-trivial delay in connection establishment. The two-minute delay wait period forced the TIME_WAIT to expire.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Testing and Analysis

This chapter discusses the execution of the benchmark tests and provides analysis of the results. Section 5.1 discusses issues related to the collection of data and Section 5.2 provides a summary of the results.

5.1 Overview

Chapter 3 and Chapter 4 discuss the methodology for designing a meaningful benchmark to measure the performance of targeted components in MYSEA. The goal is to achieve *soundness*: developing confidence that the results we derive from our measurements are well justified, with a solid understanding of the strengths and limitations of the measurement process on which we base our results [31]. We need to calibrate the test results by examining outliers and testing for consistencies. This allows us to reliably reproduce analysis results by imposing a systematic structure on the analysis process.

The "OMNI" test in netperf allows the specification of the type of output values to collect for the test. These "selectors" determine the data to be reported. Table 5.1 highlights the important selectors used. [30] provides the detailed documentation for the "selectors".

Table 5.1: Netperf "OMNI" test selectors used during benchmarking MYSEA

Selector Name	Description	Examples
PROTOCOL	Protocol used in the benchmarks.	*UDP, TCP*
DIRECTION	Direction of the data flows relative to the netperf process.	*send | recv: bidirectional request and response* *send: unidirectional request*
THROUGHPUT	Throughput for the test.	*350.0234*
THROUGHTPUT_UNITS	Units for the throughput	$10^3 bits/s$ *or* $10^6 bits/s$

5.2 Summary of Results

We compare each test scenario, relative to the scenario in which netserver is deployed on STOP without the network being protected with IPSec, treating this as a "base case". As expected, we see that the base case shows the best performance, i.e., highest throughput and lowest latency (see Figure 5.1, 5.2 and 5.3). For TCP stream performance, IPSec encryption results in a 60% decrease in throughput, whereas MYSEA processes result in between 21–24% decrease in throughput. For socket related performance, IPSec encryption results in between 24–30% increase in latency, whereas MYSEA processes result in between 93–143% increase in latency. These observations will be described more in their respective sections.

Our UDP results are inconclusive. For setups without IPSec encryption, the throughput values obtained are lower than when IPSec encryption tunnel is used. This is counter-intuitive and possibly the result of packet loss during testing. We refer the interested reader to Appendix G for the status of our UDP test results. We leave further UDP testing as future work.

5.2.1 Overhead associated with IPSec

IPSec encryption reduces unidirectional TCP transfer throughput significantly. The performance consequences of IPSec in an architecture similar to MYSEA, with server gateway and IPSec encryption of traffic between TPE and server, is a decrease in unidirectional transfer speeds, by about 60% (see Table 5.3). TCP connect latency increases an average of 24–30%, reflected in the TCP_CC, TCP_RR and TCP_CRR tests. For the details of each test, see Appendix D.

The performance of IPSec has been well studied. Shue et al [32] report performance overhead of 32–60% for Internet Key Exchange (IKE) based IPSec (Racoon uses IKE based IPSec) and provides a comprehensive study on the performance overhead of IPSec. This is similar to our results of 60% decrease in TCP throughput and socket related performance overhead range of 24–30%.

5.2.2 Overhead associated with the MYSEA Trusted Proxy

We observe a 21–25% decrease in throughput for unidirectional TCP traffic, relative to the base case (see Table 5.4), when the trusted MYSEA proxy is used for network traffic (for details, see Appendix E). We believe this decrease can be attributed to MYSEA's proxy system call design, in which the client and APS communicate via a proxy process (SSS-C) mediating all accesses, inspecting and routing TCP network packets internally before reaching the APS.

Table 5.2: Summary of the average overhead witnessed during TCP benchmarks (Figure 5.1, 5.2, 5.2)

TCP_STREAM Throughput				
	Scenario	**Mean** ($10^6 bits/s$)		**% decrease**
MYSEA	I-M F	274.95		21.67%
	I-M F-	270.57		24.67%
Base	I-M-F	351.00		-%
	I-M-F-	359.16		-%
IPSec	I M-F	139.70		60.20%
	I M-F-	144.51		59.76%
TCP_CC: Latency (ms/Trans)				
	Scenario	**Mean** ($Trans/s$)	**Latency**	**% increase**
MYSEA	I-M F	529.54	1.888	143.30%
	I-M F-	674.70	1.482	93.22%
Base	I-M F	1288.58	0.776	-%
	I-M F-	1304.01	0.767	-%
IPSec	I-M F	993.89	1.006	29.64%
	I-M F-	1020.16	0.980	27.77%
TCP_RR: Latency (ms/Trans)				
	Scenario	**Mean** ($Trans/s$)	**Latency**	**% increase**
MYSEA	I-M F	2318.23	0.431	20.40%
	I-M F-	2338.82	0.428	19.55%
Base	I-M F	2791.03	0.358	-%
	I-M F-	2792.37	0.358	-%
IPSec	I-M F	2148.25	0.465	29.89%
	I-M F-	2243.02	0.446	24.58%

The latency observed when the establishment of a new connection is involved is anywhere between 90–150%, relative to the base case (see Table 5.4), whereas TCP_RR results in a smaller increase in latency at about 20%, similar to the TCP_STREAM performance overhead. TCP_RR is similar to TCP_STREAM, with the former being a bidirectional test. The TCP_RR and TCP_STREAM results further conclude that MYSEA processes increase TCP performance overhead by about 20% as a consequence of its proxy design to route all network packets via its respective SSS-C process.

However, performance overhead increases significantly to 90–150% when the establishment of

Table 5.3: Summary of IPSec results

TCP_STREAM Throughput			
	Scenario	**Mean** ($10^6 bits/s$)	**% decrease**
F-	I-M-F- I M-F-	359.16 144.51	60.00%
F	I-M-F I M-F	351.00 139.70	60.20%
TCP_CC: Latency (ms/Trans)			
	Scenario	**Mean** ($Trans/s$)	**Latency**	**% increase**
F-	I-M-F- I M-F-	1304.01 1020.16	0.767 0.980	27.77%
F	I-M-F I M-F	1288.58 993.89	0.776 1.006	29.64%
TCP_RR: Latency (ms/Trans)			
	Scenario	**Mean** ($Trans/s$)	**Latency**	**% increase**
F-	I-M-F- I M-F-	2792.37 2243.02	0.358 0.446	24.58%
F	I-M-F I M-F	2791.03 2148.25	0.358 0.465	29.89%
TCP_CRR: Latency (ms/Trans)			
	Scenario	**Mean** ($Trans/s$)	**Latency**	**% increase**
F-	I-M-F- I M-F-	1183.12 903.96	0.845 1.106	30.89%
F	I-M-F I M-F	1152.27 889.10	0.868 1.125	29.61%

a new connection is involved. We believe this can be attributed to the creation of a new SSS-C process for every new TCP connection, resulting in significant performance overhead.

5.2.3 MYSEA Federation Overhead

When a MYSEA service (APS) and the TPS process are deployed on different servers, the APS's proxy causes a Federated request to fetch information about the client's current level from the MYSEA server hosting the TPS. In practice, these security checks occur per connection (for UDP, per packet). In our testing, the netserver process acts as an APS, allowing us to interpret the overhead associated with the intra-Federation communication.

Table 5.4: Summary of MYSEA results

TCP_STREAM Throughput				
	Scenario	**Mean** ($10^6 bits/s$)		**% decrease**
F-	I-M-F- I-M F-	359.16 270.57		24.67%
F	I-M-F I M-F	351.00 274.95		21.67%
TCP_CC: Latency (ms/Trans)				
	Scenario	**Mean** ($Trans/s$)	**Latency**	**% increase**
F-	I-M-F- I M-F-	1304.01 674.70	0.767 1.482	93.22%
F	I-M-F I M-F	1288.58 529.54	0.776 1.888	143.30%
TCP_RR: Latency (ms/Trans)				
	Scenario	**Mean** ($Trans/s$)	**Latency**	**% increase**
F-	I-M-F- I M-F-	2792.37 2338.82	0.358 0.428	19.55%
F	I-M-F I M-F	2791.03 2318.23	0.358 0.431	20.39%
TCP_CRR: Latency (ms/Trans)				
	Scenario	**Mean** ($Trans/s$)	**Latency**	**% increase**
F-	I-M-F- I M-F-	1183.12 575.88	0.845 1.736	105.44%
F	I-M-F I M-F	1152.27 467.30	0.868 2.140	146.54%

For TCP services, we observe a greater latency when the APS and TPS are deployed on different servers. From the results of the TCP_CC and TCP_CRR tests, setup "F" shows a significant increase in latency of about 50% compared to setup "F-". We believe this can be attributed to the additional performance overhead associated with the MYSEA processes communicating internally when the APS and TPS are on different servers.

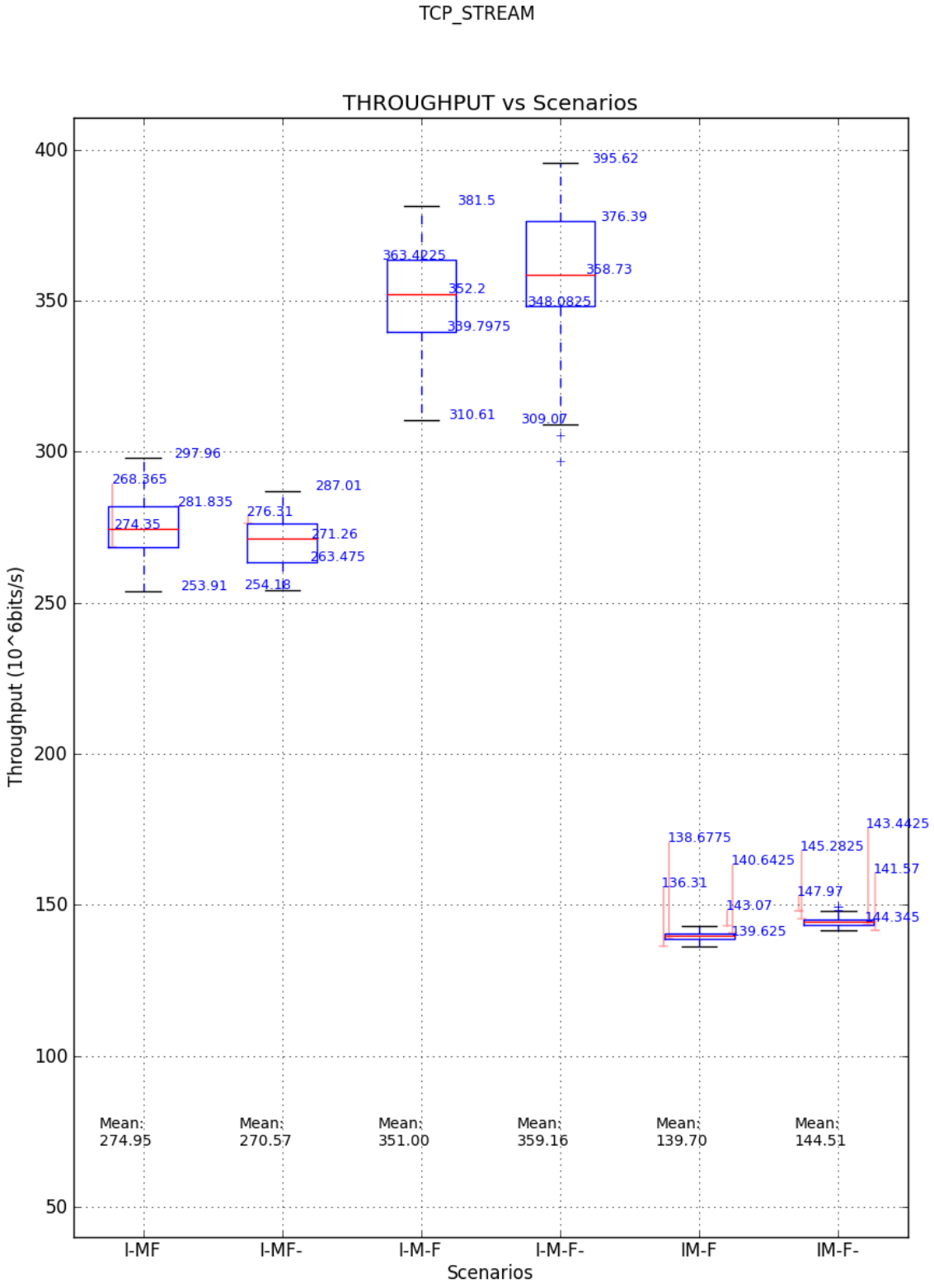

Figure 5.1: Boxplot showing the summary of TCP_STREAM performance for the six setups

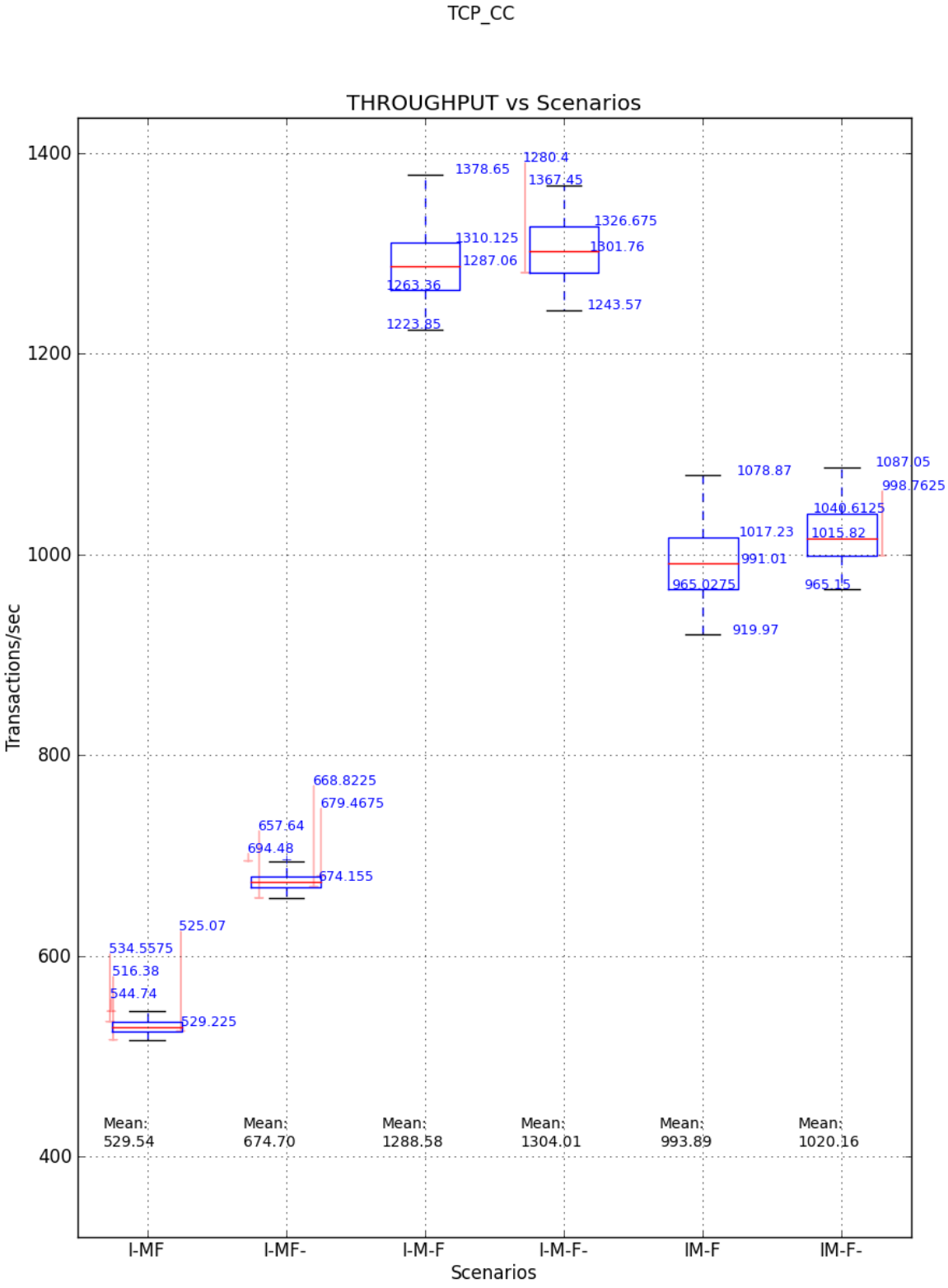

Figure 5.2: Boxplot showing the summary of TCP_CC socket performance for the six setups

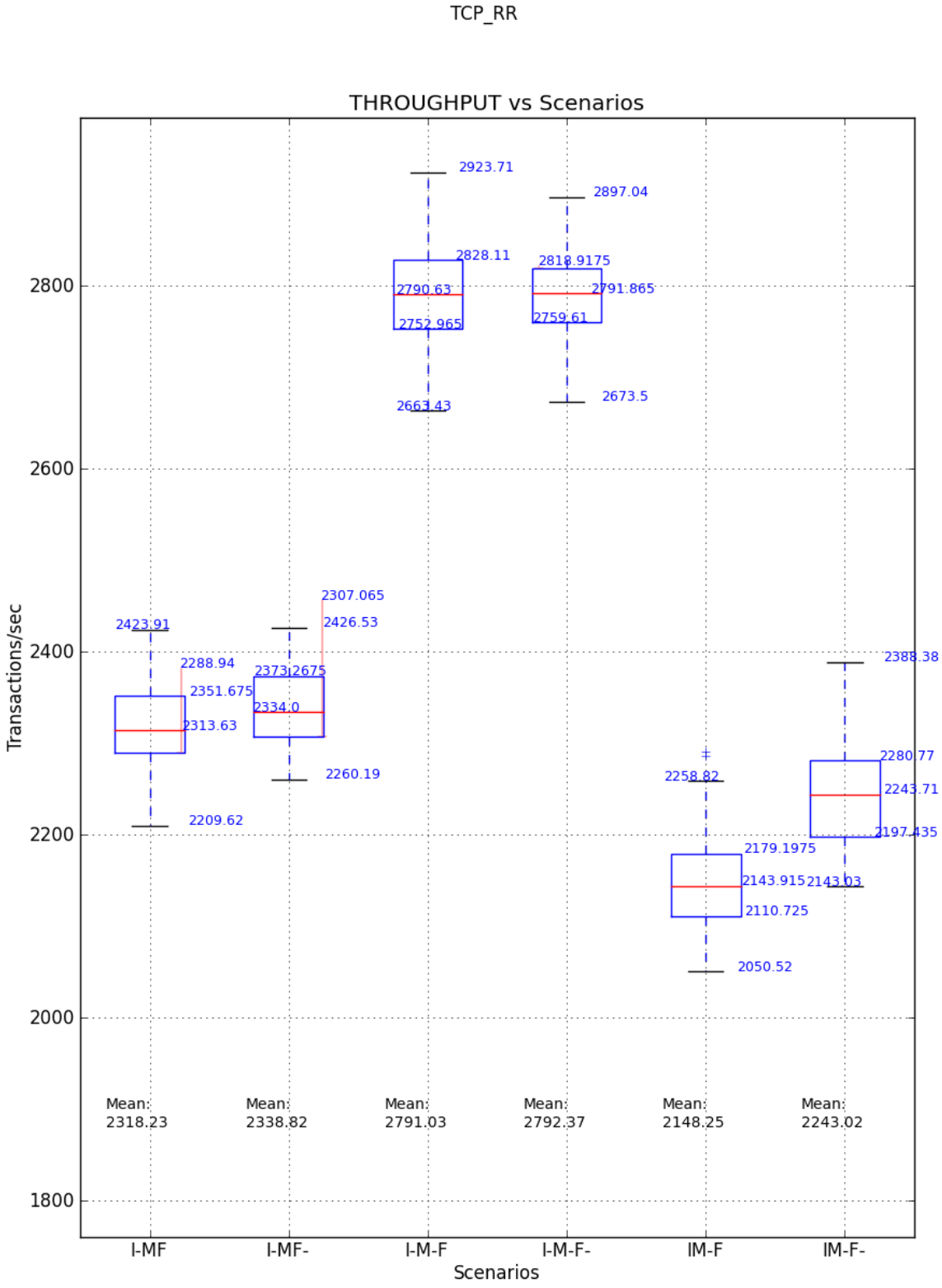

Figure 5.3: Boxplot showing the summary of TCP_RR request and response performance for the six setups

CHAPTER 6:
Conclusion

We analyzed the architecture of the MYSEA framework, and identified the major components and services that create a high assurance MLS environment. Overall, we observe that the MYSEA trusted proxy architecture incurs significant overhead (impacting throughput and latency for network connections). On average, the latency for new connections more than doubles (about 118%) compared to MYSEA's underlying platform. These costs are associated with connection setup, i.e. starting a socket proxy on behalf of the user. Once a connection is established, we observe an average throughput decrease of about 23% for unidirectional TCP streams, associated with the cost of proxying a socket write() request. In a Federated setting, the way state is communicated among servers alone is associated with an increase in latency of about 50%.

6.1 Recommendations for Future Work

This thesis performed a high level analysis on the performance of MYSEA, focusing on the comparison of the mean latency values and throughput. Further detailed mathematical analysis can be performed on the data, finding other trends for the current set of data. The UDP results are not conclusive, and future work can look into the possible causes and mitigations to study the performance overhead of UDP packets. Future work can analyze other components and services in MYSEA, such as the performance of the TPE and server gateway. Future work can explore recommendations for improving the performance of the bottlenecks identified. Future work can also measure the performance of MYSEA in setups "I M F-" and "I M F", which are closer comparisons to the operational environment that MYSEA is deployed in.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A:
Netserver Modifications

This appendix describes the modifications to netserver to operate in the MYSEA environment.

1. **Include MYSEA headers**
   In the header file: netlib.h, insert the following include statement.
   `#include "mysea_portability.h"`
2. **Insert flags for different compilation modes**
   Flags are used for the compilation of netserver in its two different modes of operation: *inetd* and *daemon*. *inetd* mode is used in the MYSEA environment, under scenario “M", while *daemon* mode is used in STOP, under scenario “M-". We need to compile the netserver source code with support for MYSEA processes in scenario “M". Any modifications made to the source code to operate in MYSEA environment use flags inserted on the command line to obtain the appropriate compilation.

```
#ifdef USE_M_SOCKET
#include "mysea_portability.h"
#endif
```

Figure A.1: Insertion of MYSEA header with flags to compile for MYSEA support

3. **Insert necessary code**
   Additional code and functions are added to netserver to integrate with MYSEA processes. The following functions and headers are added to the source code:

```
#ifdef USE_M_SOCKET
int my_fd = -1;
#endif


void exit_mysea(void)
{
#ifdef USE_M_SOCKET
    mport_exit(0);
#endif // USE_M_SOCKET
    exit(0);
}
```

Figure A.2: Addition of headers to source code in file: netserver.c

```
void
set_mysea_server_sock(int mskt_fd) {
#ifdef WIN32
  server_sock = (SOCKET)GetStdHandle(STD_INPUT_HANDLE);
#elif !defined(__VMS)
  if (server_sock != INVALID_SOCKET) {
    printf("yo, Iz ain't invalid!\n");
    exit(1);
  }
  server_sock = mskt_fd;
#else
  printf("\tALL ELSE\n");
  if ((server_sock = mskt_fd) == INVALID_SOCKET) {
    fprintf(stderr,
            "%s: failed to grab aux server socket: %s (errno %s)\n",
            __FUNCTION__,
            strerror(errno),
            errno);
    fflush(stderr);
    exit(1);
  }
#endif
}
```

Figure A.3: Addition of functions to source code in file: netserver.c, function: main

```
int _cdecl
main(int argc, char *argv[]) {
#ifdef USE_M_SOCKET
  int mysea_exit = 0;
  int result;
  // Initialize underlying MYSEA modules
  result = mport_init(APS_MODE, "netserver_log", NULL);
    if (result != MPORT_OK) {
      result = -1;
      printf("mport_init error\n");
      mysea_exit = 1;
    }else{ // find the MSKT fd
      result = mport_get_fd(&my_fd);
      if (result != MPORT_OK){
        my_fd = -1;
        printf("mport_get_fd error\n");
        mysea_exit = 1;
      }else{
        result = mport_get_file_stream(&where);
        if (result != MPORT_OK){
          printf("mport_get_file_stream error");
          mysea_exit = 1;
        }else{
          fprintf(where,"Using MSKT fd [%d]\n", my_fd);
        }
      }
    }
    if (mysea_exit){
      printf("Initizalition error, exiting");
      mport_exit(1);
    }
  atexit(&exit_mysea);
  /*result stores the mskt_fd */
#endif     // USE_M_SOCKET
#ifdef WIN32
```

Figure A.4: Modifications to source code in file: netserver.c, function: main

```
#ifdef USE_M_SOCKET
  set_mysea_server_sock(my_fd);
  /*open_debug_file();*/
  process_requests();
#else
  if (child) {
    /* we are the child of either an inetd or parent netserver via
     spawning (Windows) rather than fork()ing. if we were fork()ed
     we would not be coming through this way. set_server_sock() must
     be called before open_debug_file() or there is a chance that
     we'll toast the descriptor when we do not wish it. */
    set_server_sock();
    open_debug_file();
    process_requests();
  }
  else if (daemon_parent) {
    /* we are the parent daemonized netserver
     process. accept_connections() will decide if we want to spawn a
     child process */
    accept_connections();
  }
  else {
    /* we are the top netserver process, so we have to create the
     listen endpoint(s) and decide if we want to daemonize */
    setup_listens(local_host_name,listen_port,local_address_family);
    if (want_daemonize) {
      daemonize();
    }
    accept_connections();
  }
#endif
```

Figure A.5: Modifications to source code in file: netserver.c, function: main

APPENDIX B:
Installing and Running the Benchmark Tool

Two files are necessary for the installation and execution of Netperf benchmark tool: **stop.v1.0.tar** and **linux.v1.0.tar**.

stop.v1.0.tar is used for the installation of netserver process on STOP server, while **linux.v1.0.tar** is used for the installation of netperf client and execution of benchmark tests on the TPE. Both are under configuration management within the MYSEA project.

B.1 Installing Netserver

1. **Decompress stop.v1.0.tar**

```
tar -xf stop.v1.0.tar
```

2. **Select mode of installation**
   Inetd is used for the setup “M" while *daemon* is used for the setup “M-". The default mode is *inetd* mode.

inetd mode:	`./mysea_install_netserver.sh -m inetd`
daemon mode:	`./mysea_install_netserver.sh -m daemon`

3. **Optional**
   Specify the control port number for netserver to monitor on the network. The default is 12865.

```
./mysea_install_netserver.sh -cp 10000
```

   Specify the *help* flag “h" for more options.

```
./mysea_install_netserver.sh -h
```

B.2 Installing Netperf

1. **Decompress linux.v1.0.tar**

```
tar -xf linux.v1.0.tar
```

2. **Optional**
   Specify the *help* flag “h" for more options.

```
./mysea_install_netperf.sh -h
```

B.3 Execute benchmark tests

1. **Installing Netserver and Netperf**
   Refer to Section B.1 and Section B.2 for the installation details
2. **Execute benchmark tests**
   Provide the necessary flags for the execution of different tests.

./mysea_run_netperf.sh	
Flags	
-cp|conPort	Control port that netserver is monitoring
-dp|dataPort	Data port used for transmission of data between Netserver and Netperf
-H |host	IP address or hostname of the targeted machine, where netserver server process is residing
-n |numTests	Number of times each test is executed
-t |test	Name of the test to be executed
-tl|testlen	The length of time in seconds each test is executed.

3. **Collect Results**
   Netperf collects the results from the tests and places them in the current directory. The default format of the output file is:

```
result-[test]-[testlen]-[numTests]-yyyymmddHHMMSS
```

4. **Optional**

   Specify the *help* flag “h" for more options.

```
./mysea_run_netperf.sh -h
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C:
Overview of Results

This appendix contains the boxplot summaries of all the testing. The boxplots compare the results of the benchmark tests executed under all the scenarios.

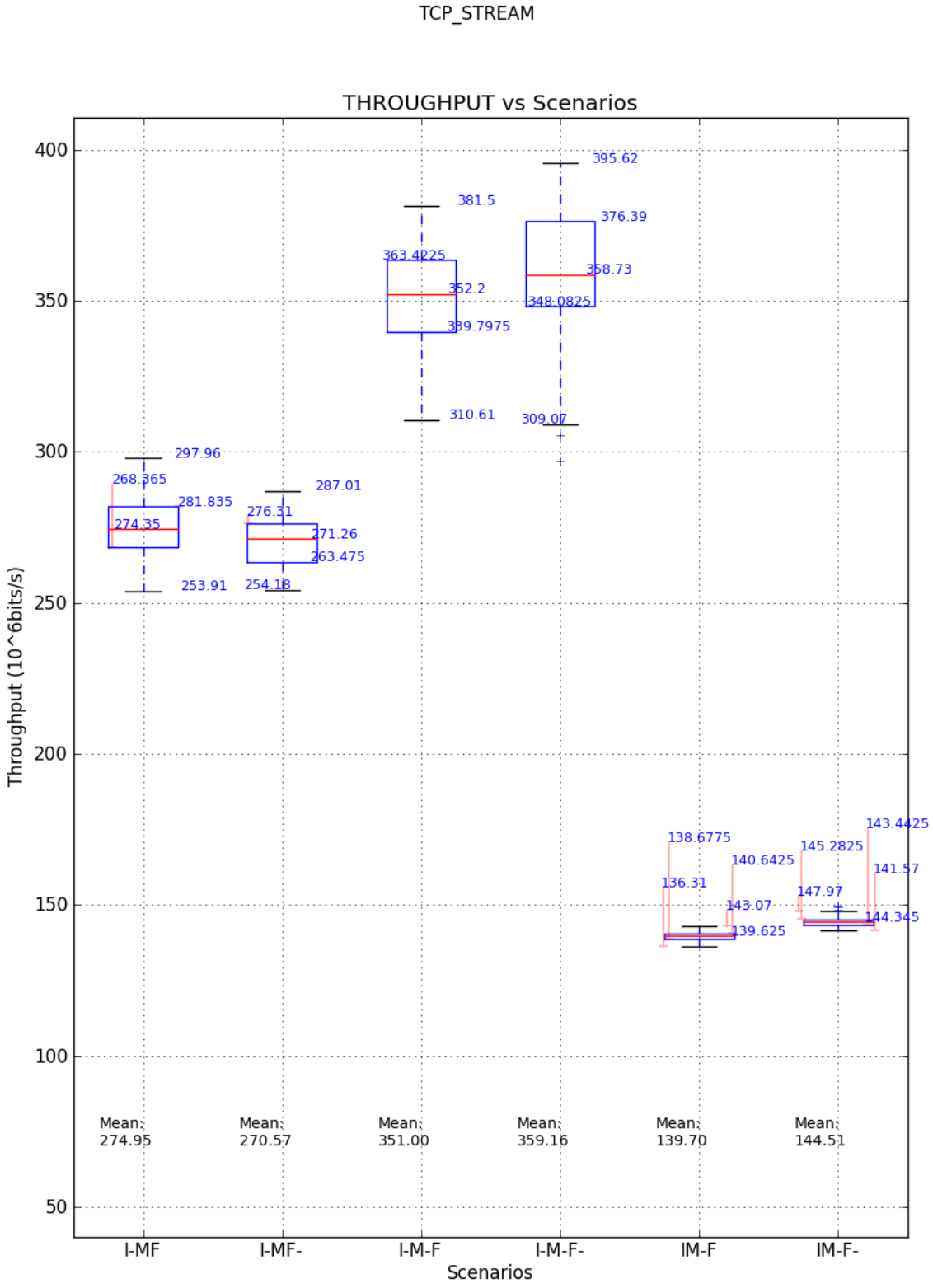

Figure C.1: Boxplots of TCP_STREAM Tests of all scenarios

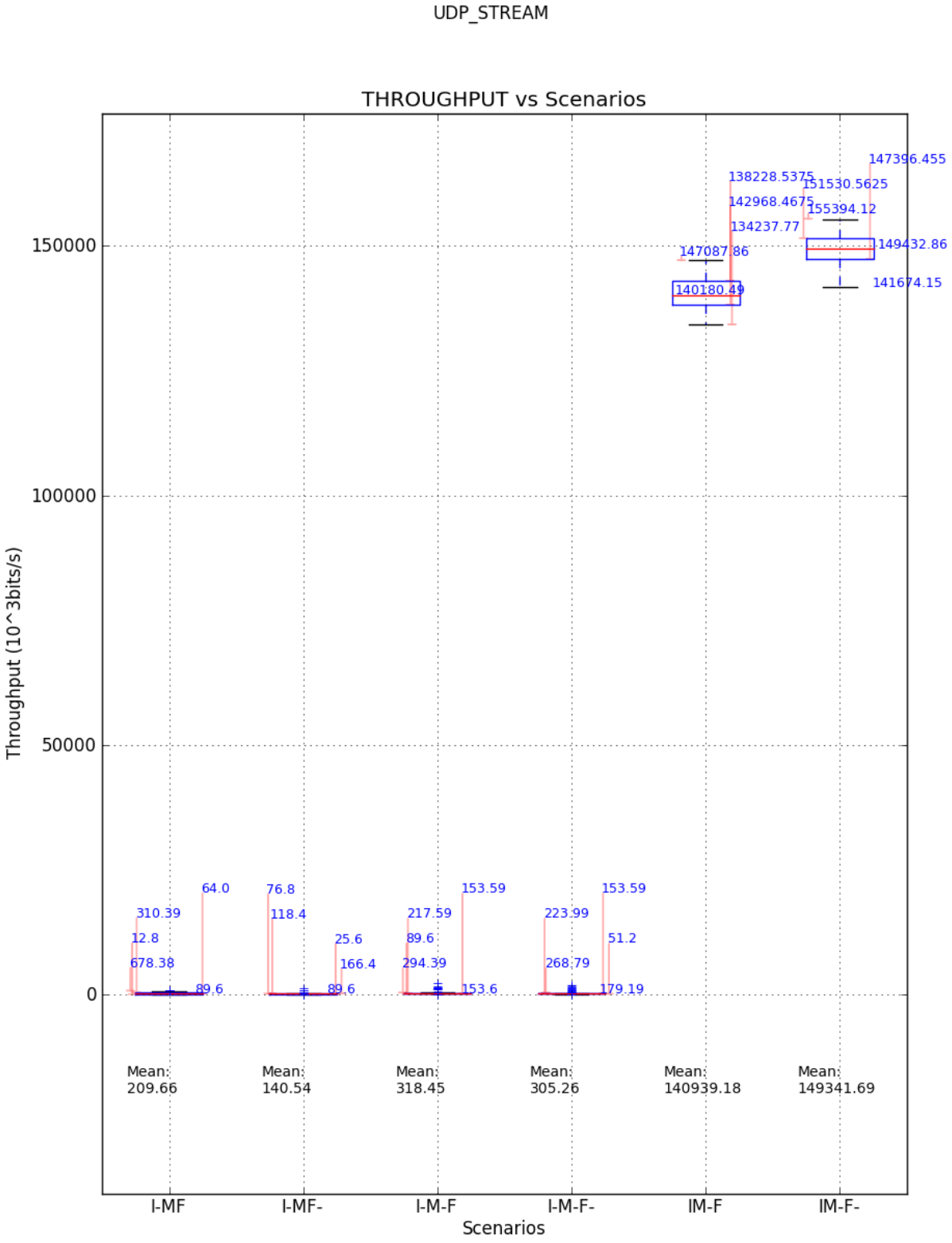

Figure C.2: Boxplots of UDP_STREAM Tests of all scenarios

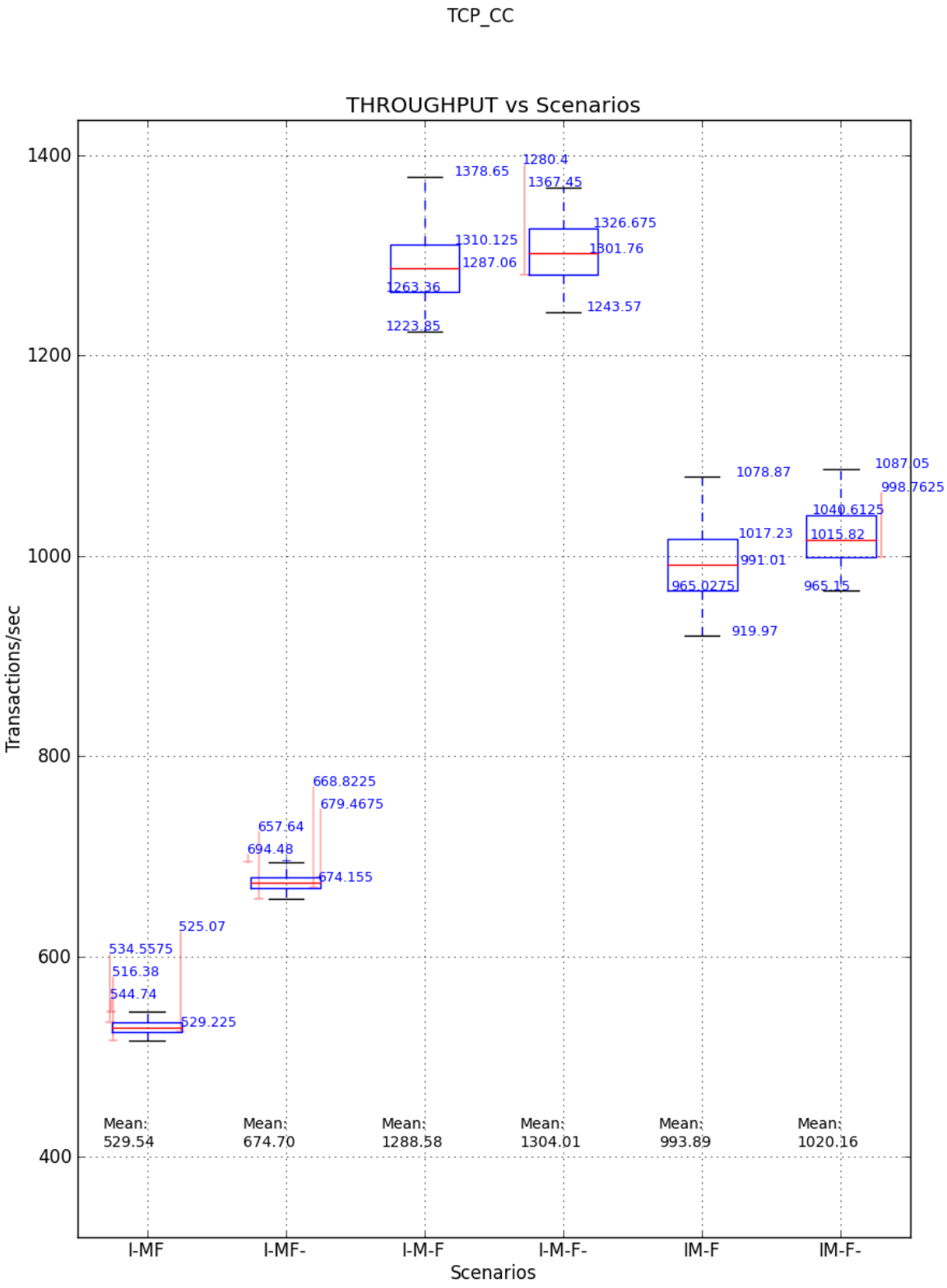

Figure C.3: Boxplots of TCP_CC Tests of all scenarios

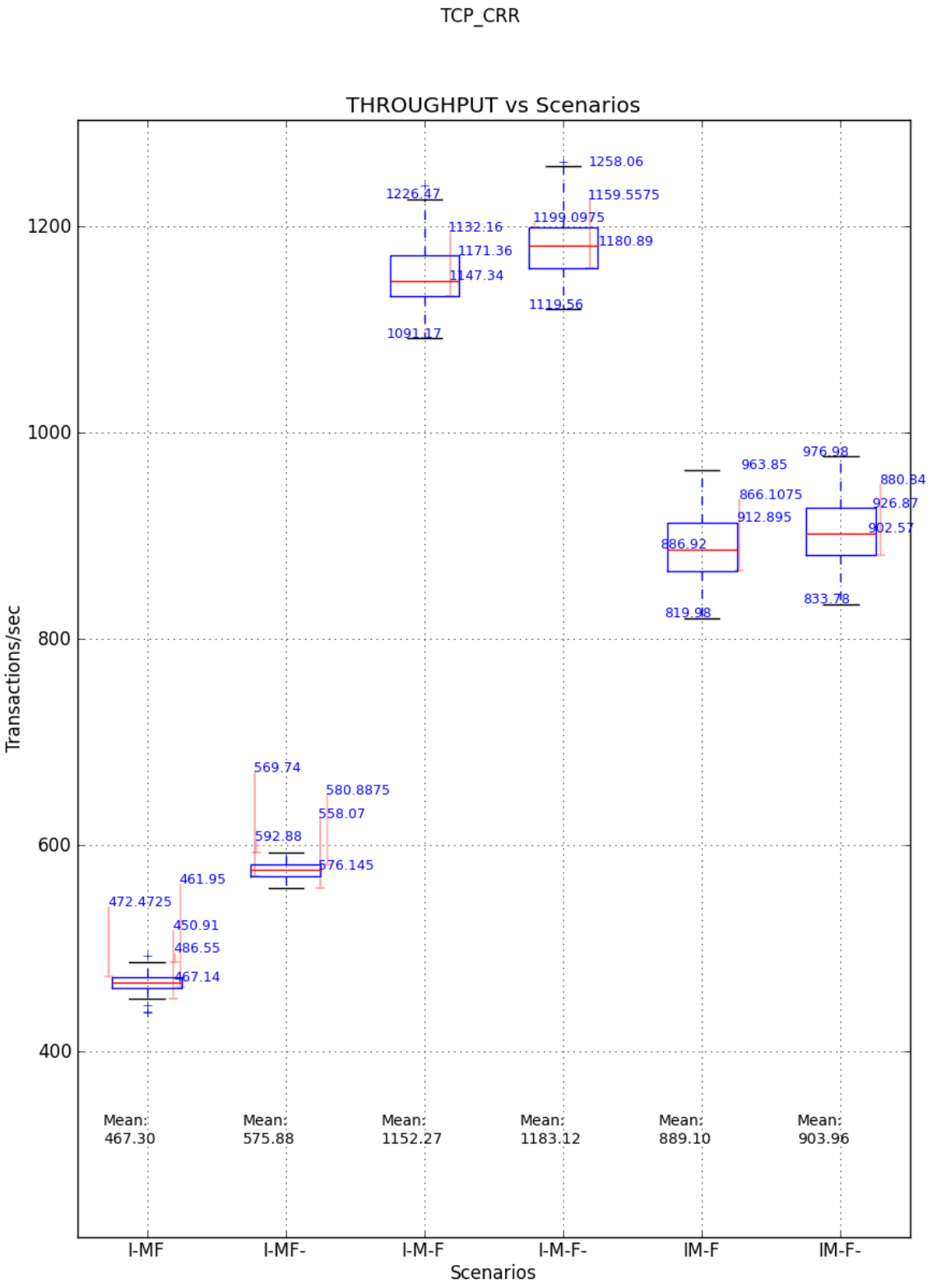

Figure C.4: Boxplots of TCP_CRR Tests of all scenarios

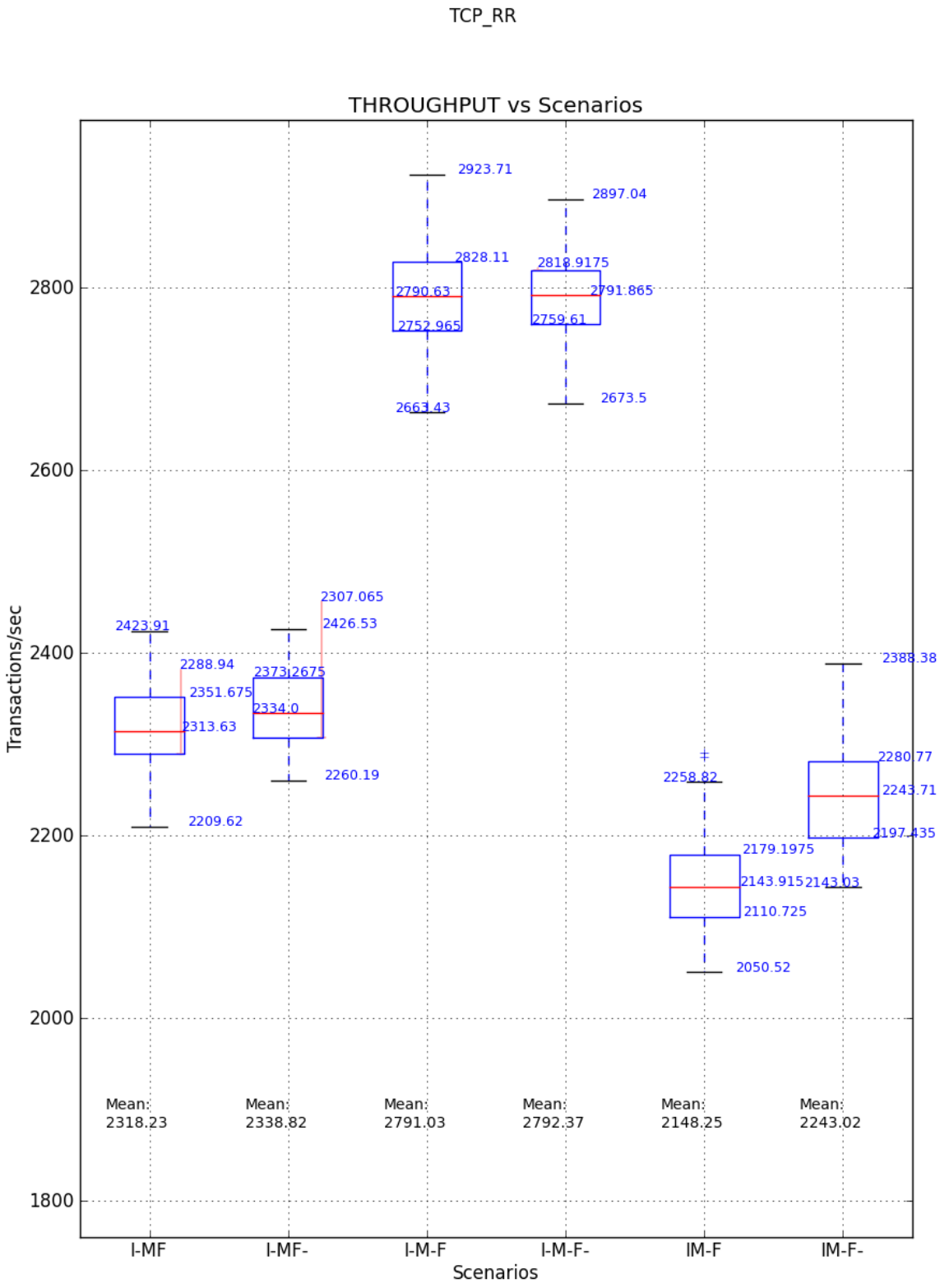

Figure C.5: Boxplots of TCP_RR Tests of all scenarios

APPENDIX D:
Results of IPSec Comparison

This appendix contains the results of the benchmark tests used for the analysis of the performance overhead of IPSec.

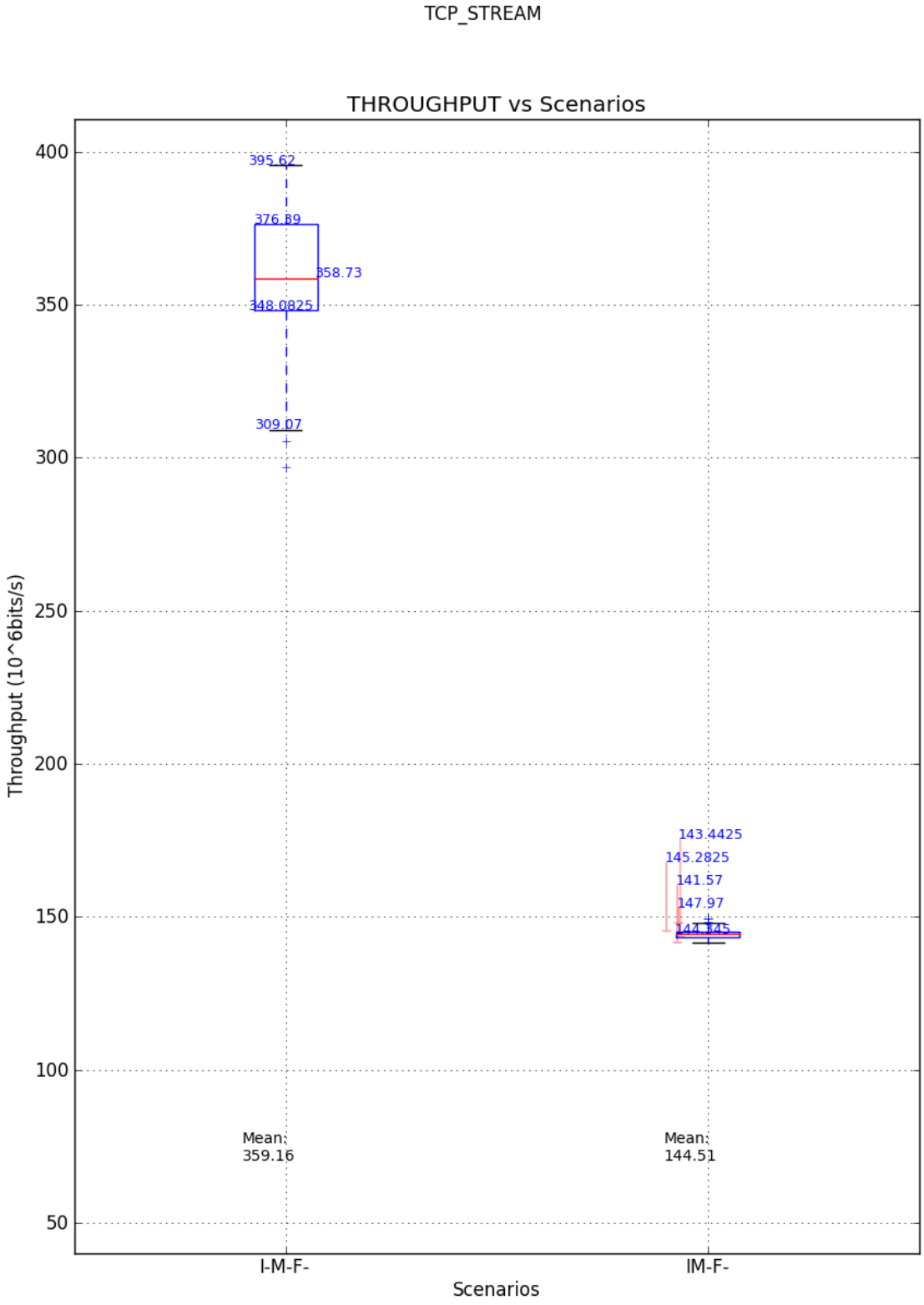

Figure D.1: Boxplots comparison of IPSec using TCP_STREAM Tests

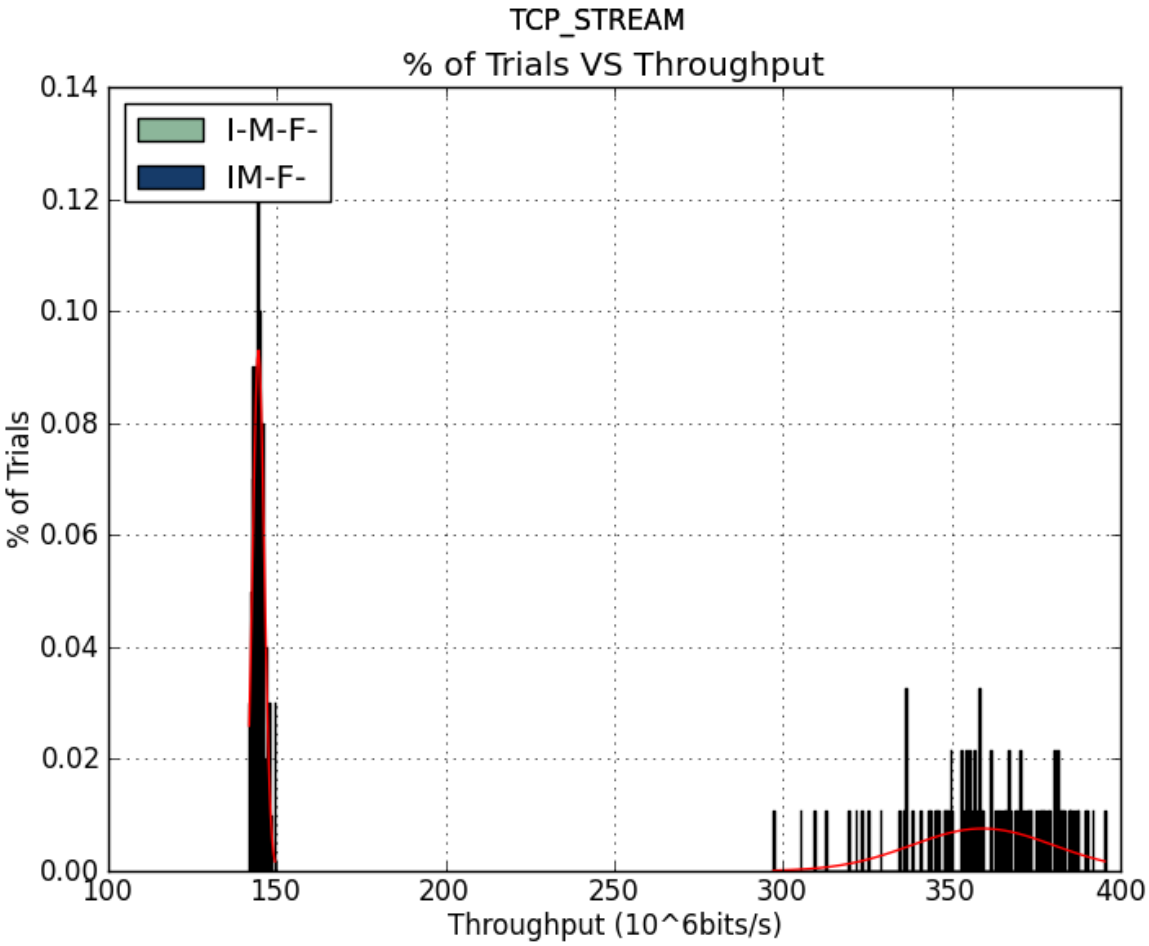

Figure D.2: Histogram comparison of IPSec using TCP_STREAM Tests

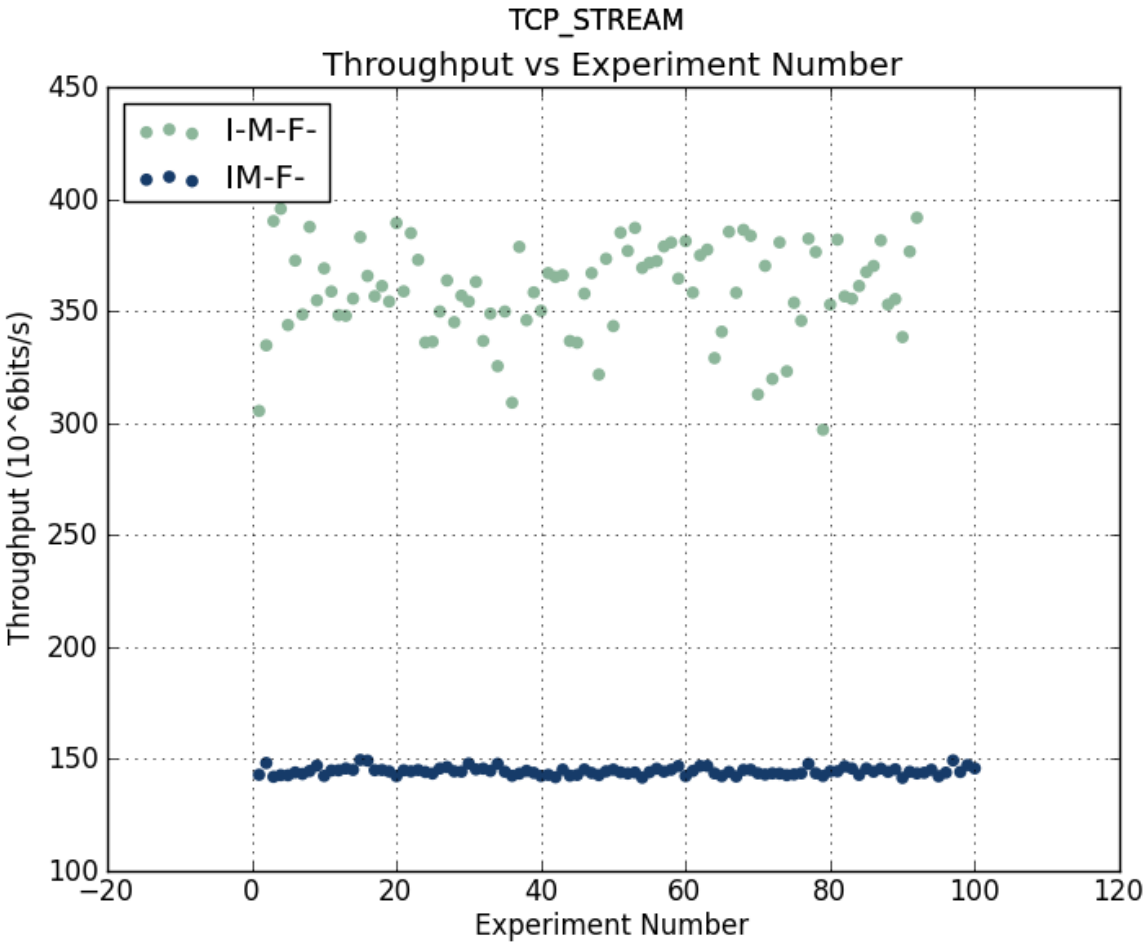

Figure D.3: Scatterplot comparison of IPSec using TCP_STREAM Tests

Figure D.4: Boxplots comparison of the performance of IPSec using UDP_STREAM Tests

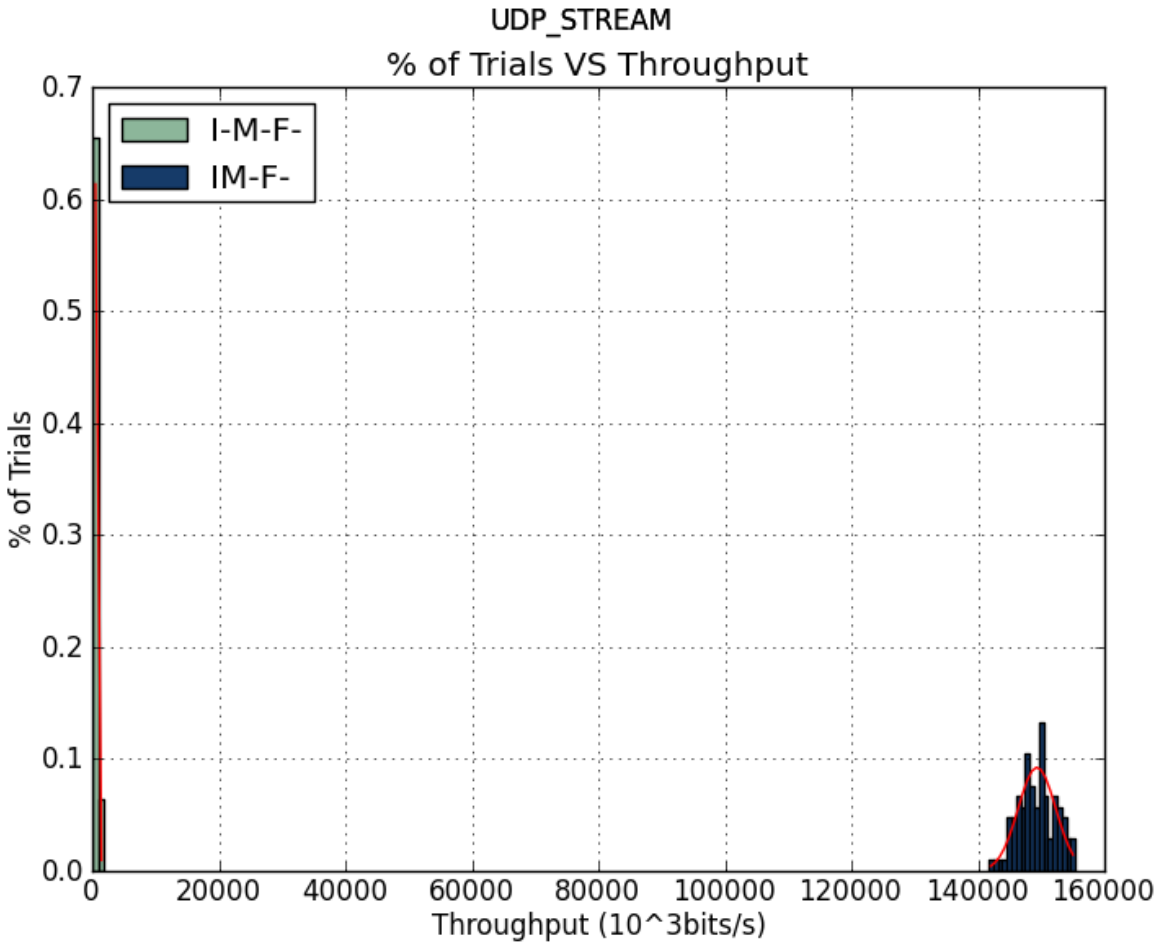

Figure D.5: Histogram comparison of the performance of IPSec using UDP_STREAM Tests

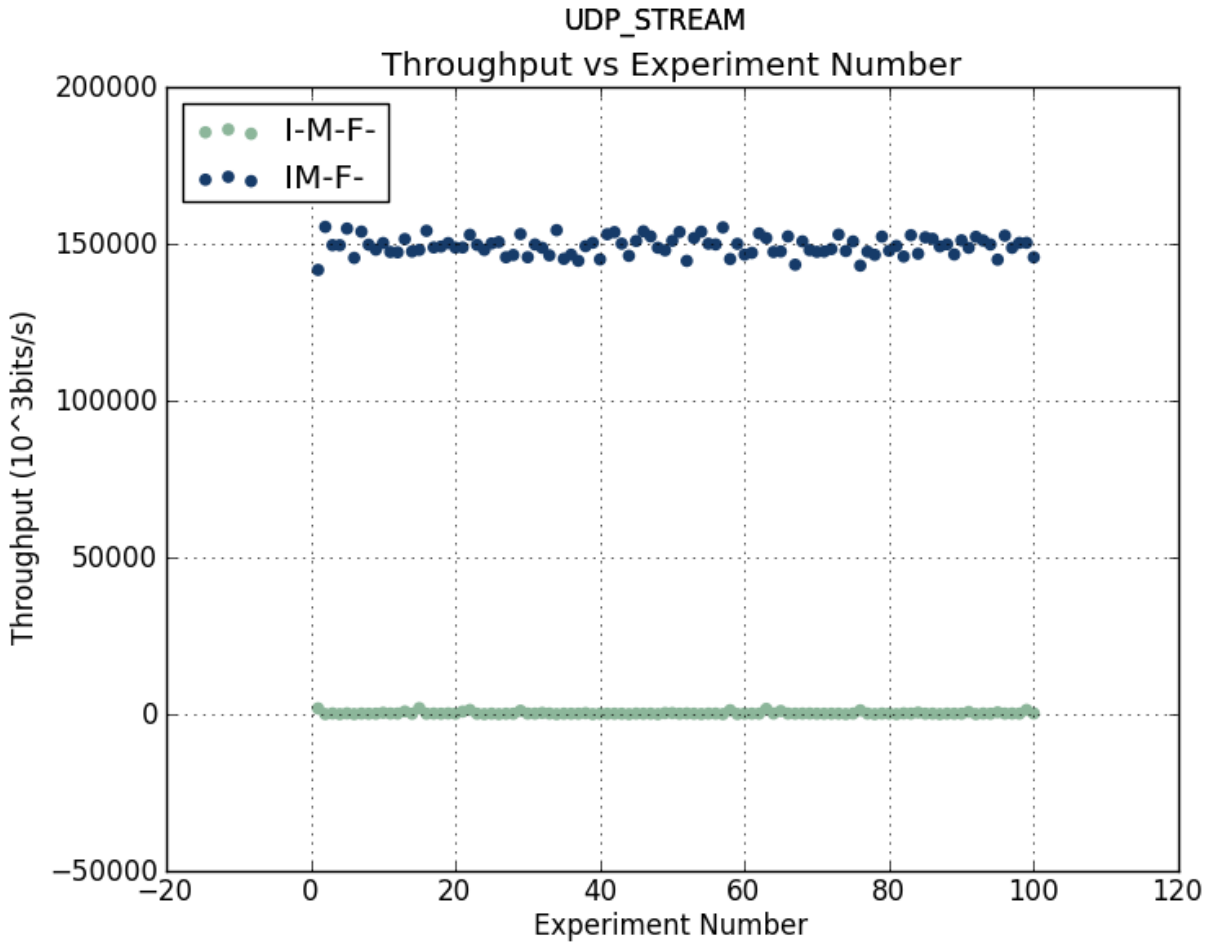

Figure D.6: Scatterplot comparison of the performance of IPSec using UDP_STREAM Tests

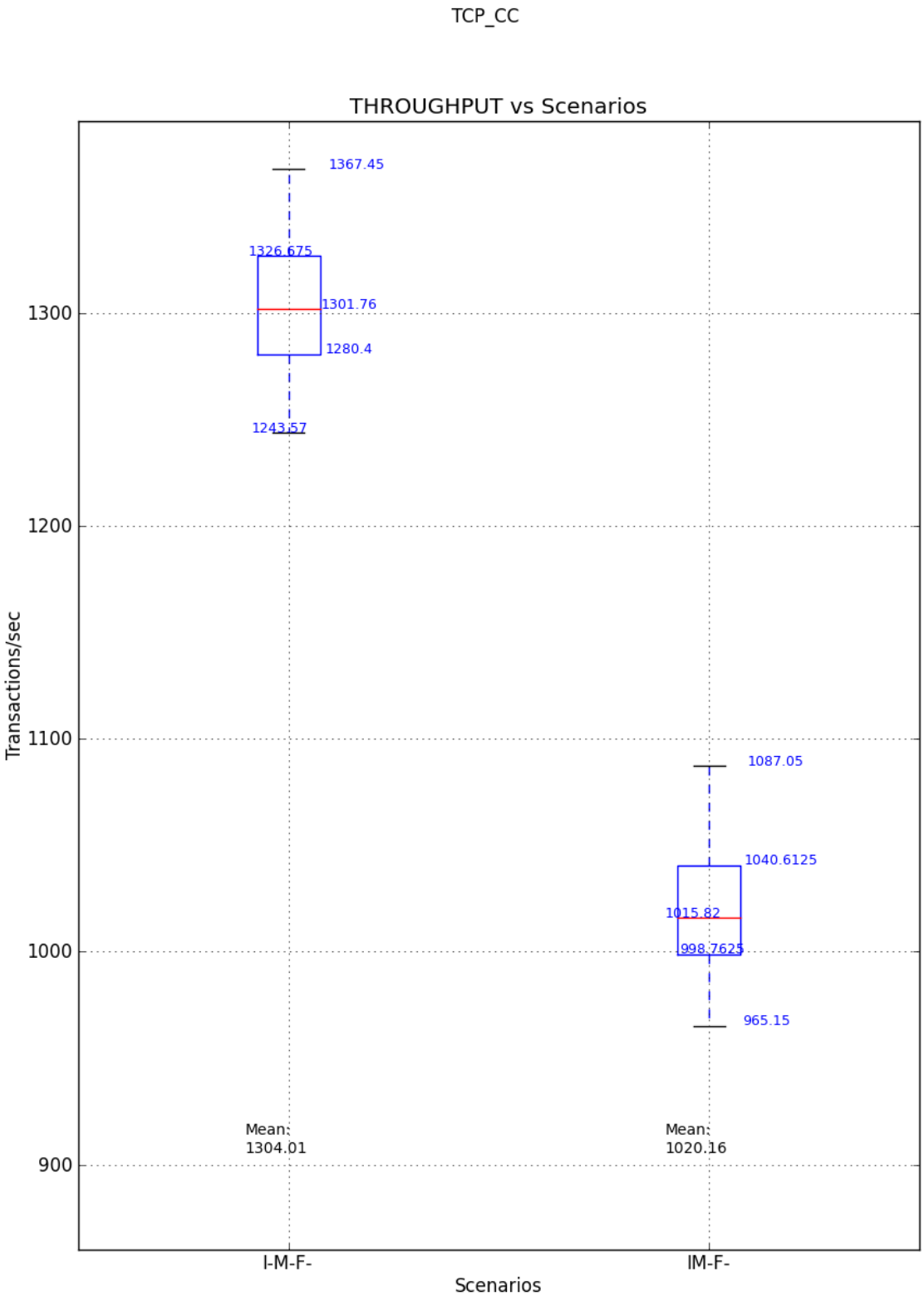

Figure D.7: Boxplots comparison of the performance of IPSec using TCP_CC Tests

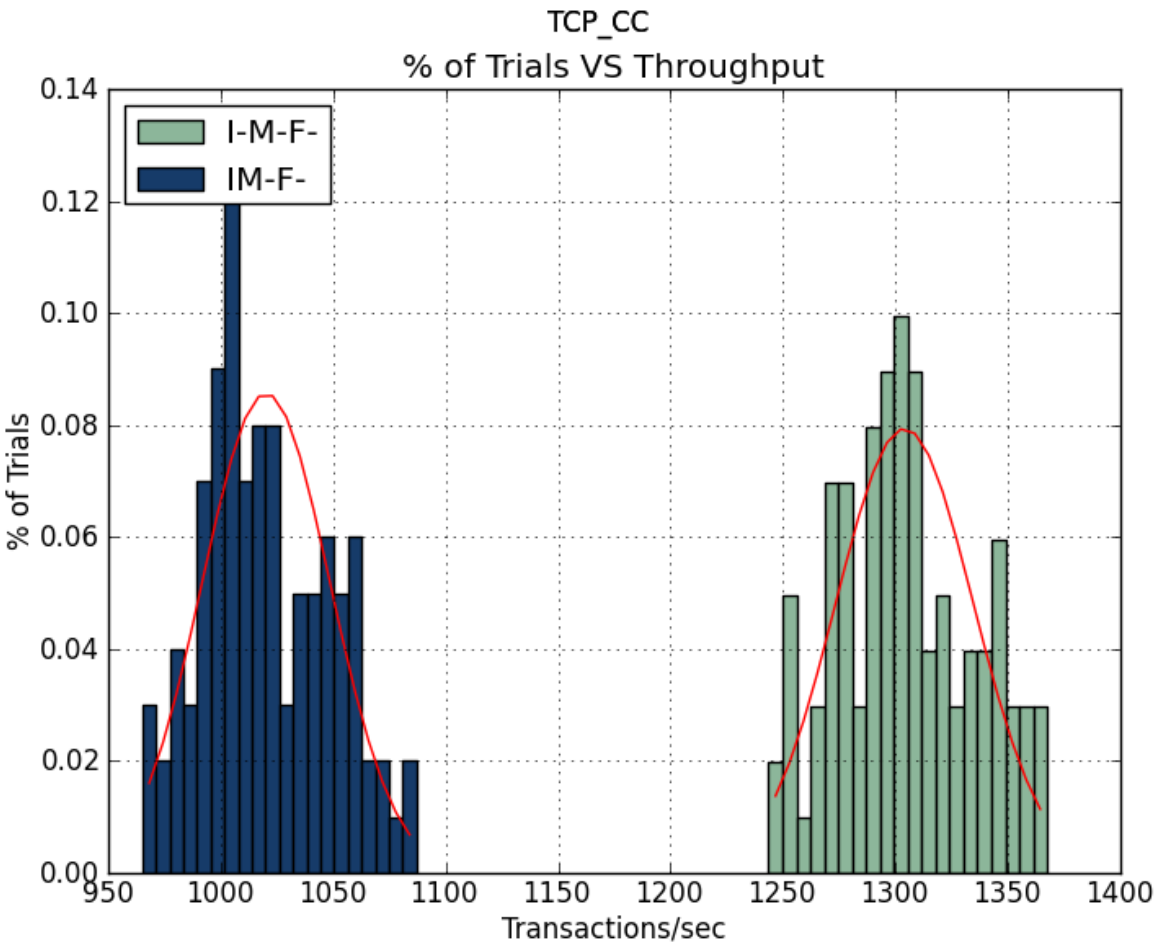

Figure D.8: Histogram comparison of the performance of IPSec using TCP_CC Tests

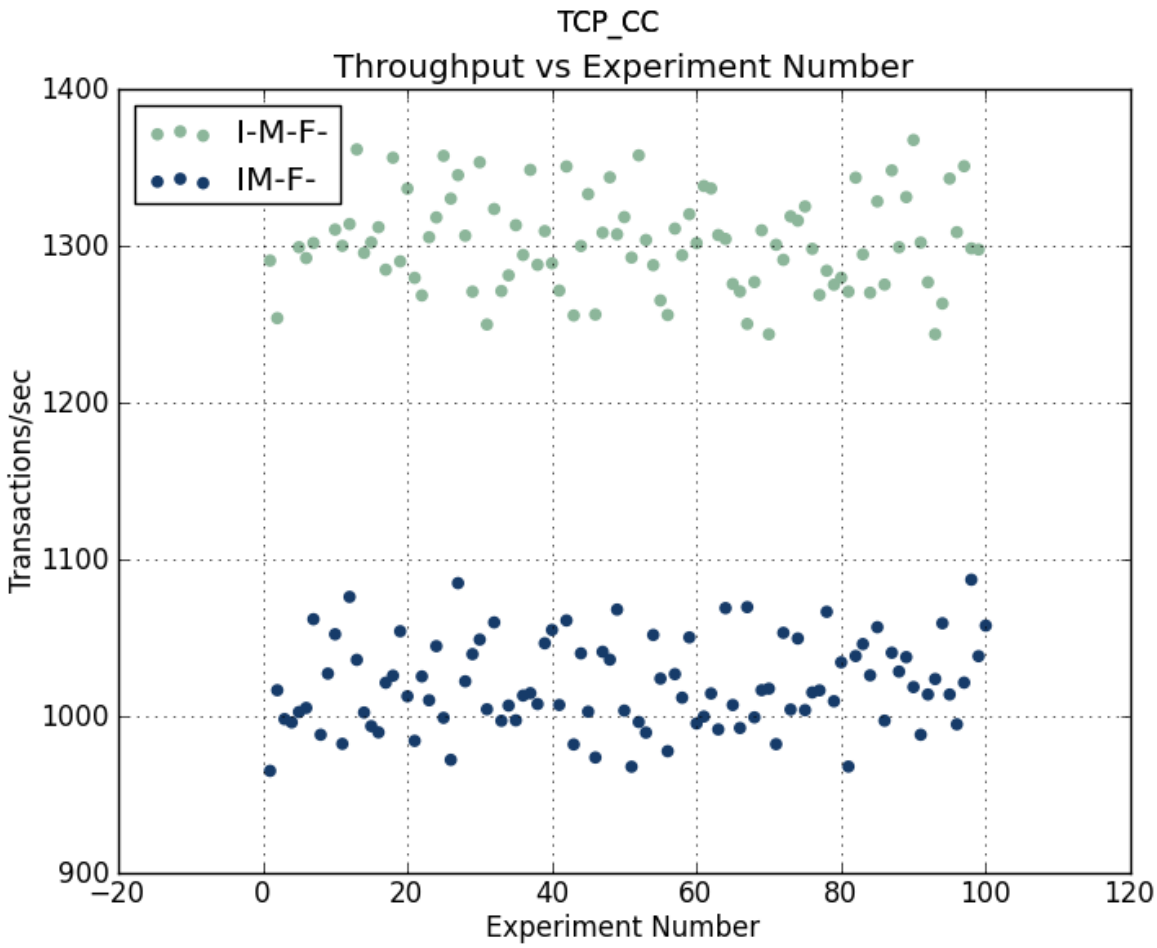

Figure D.9: Scatterplot comparison of the performance of IPSec using TCP_CC Tests

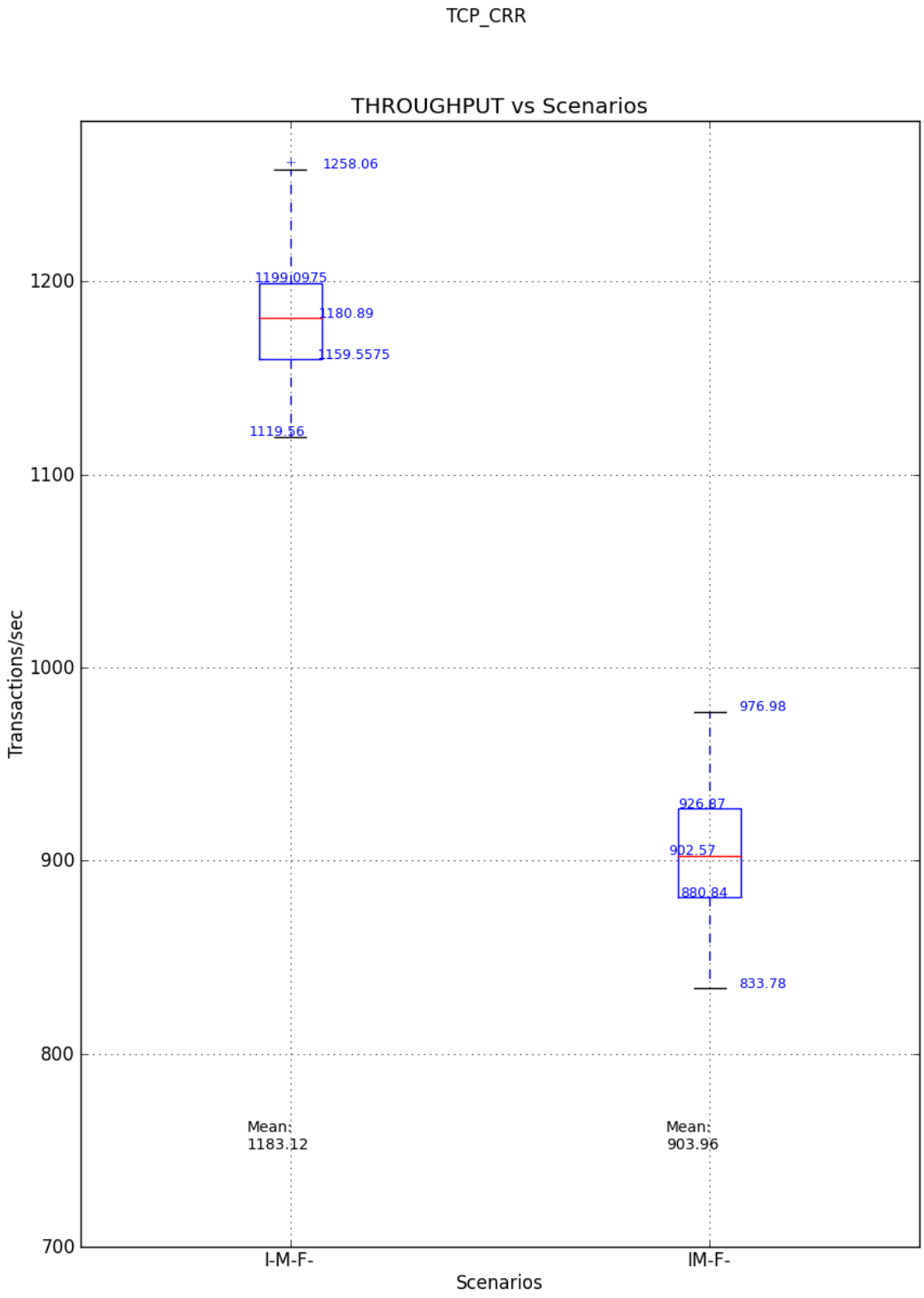

Figure D.10: Boxplots comparison of the performance of IPSec using TCP_CRR Tests

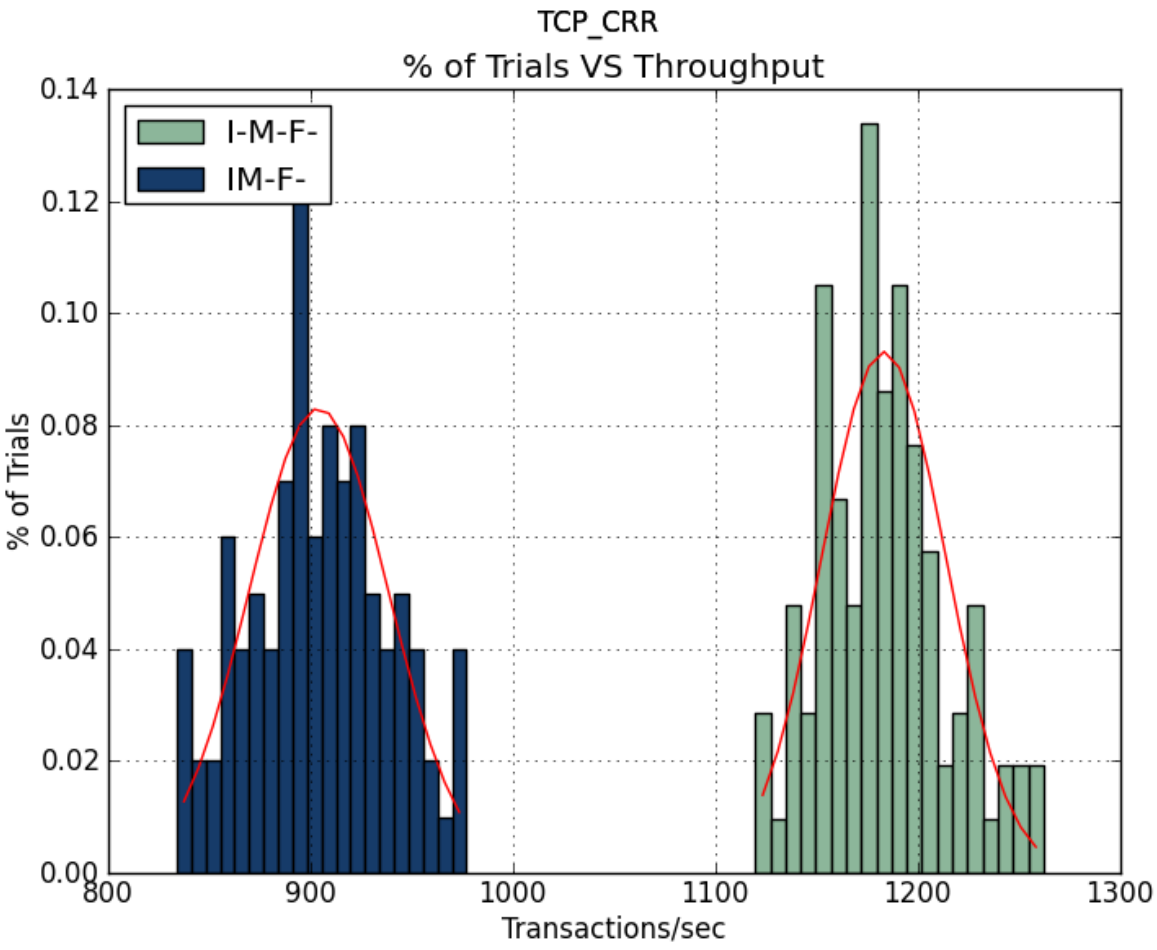

Figure D.11: Histogram comparison of the performance of IPSec using TCP_CRR Tests

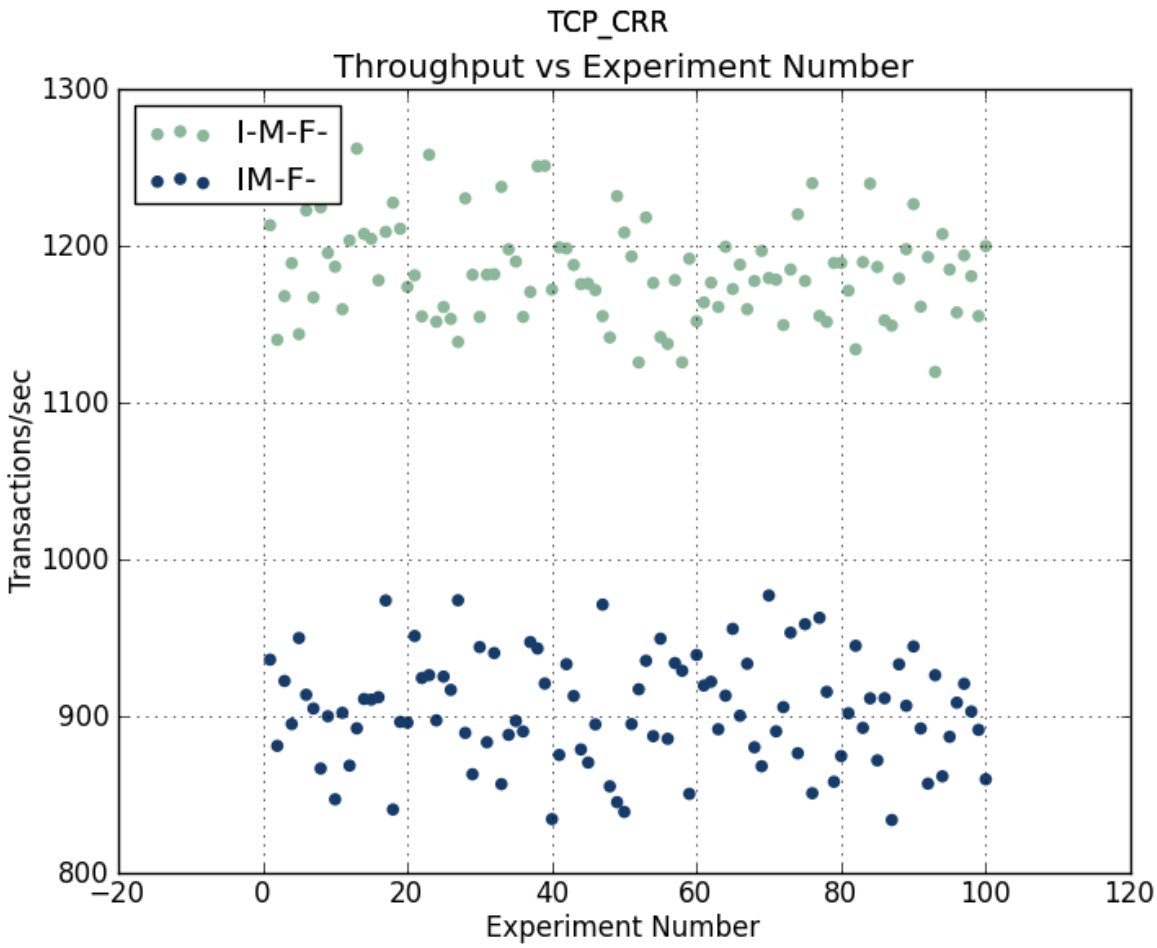

Figure D.12: Scatterplot comparison of the performance of IPSec using TCP_CRR Tests

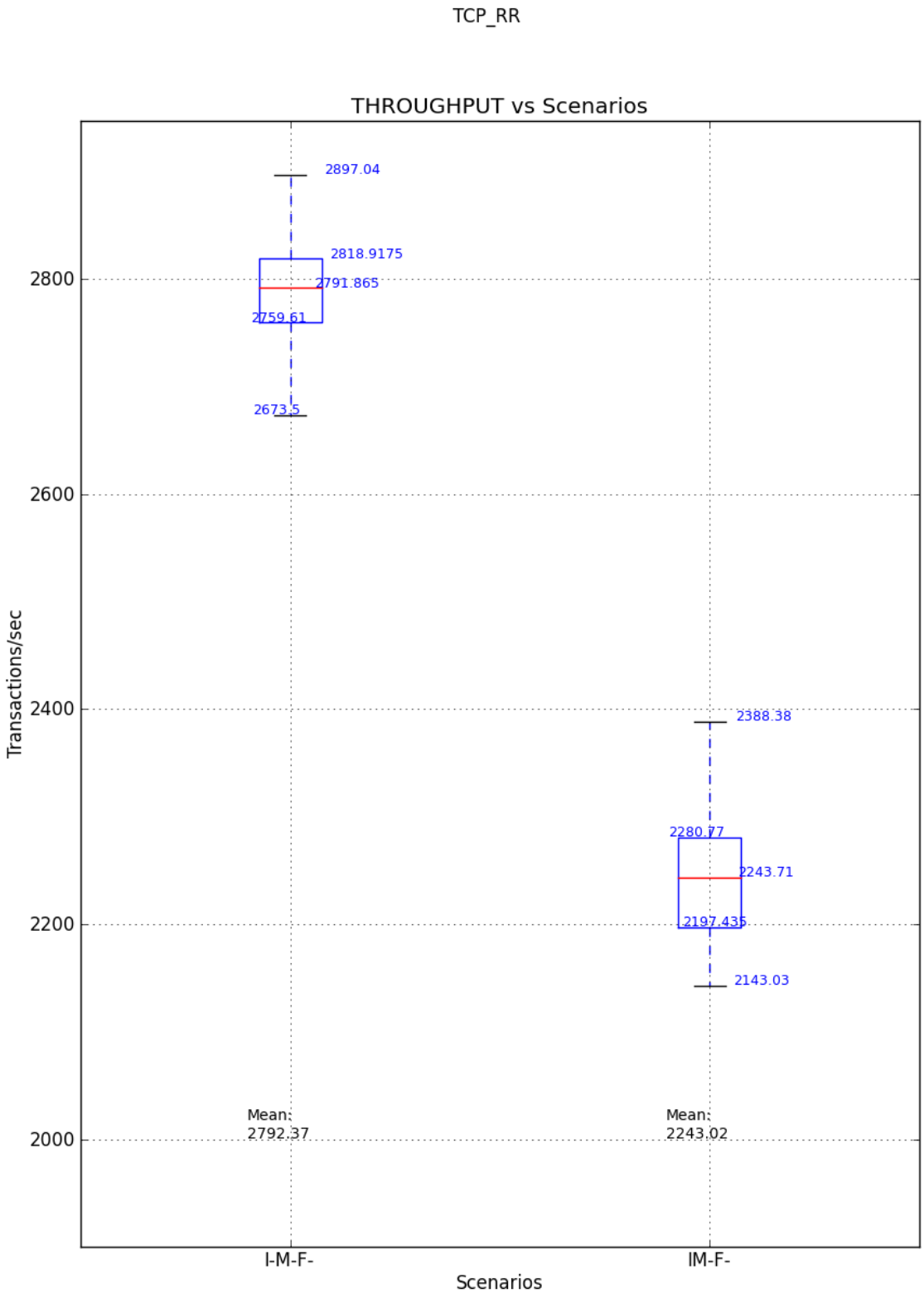

Figure D.13: Boxplots comparison of the performance of IPSec using TCP_RR Tests

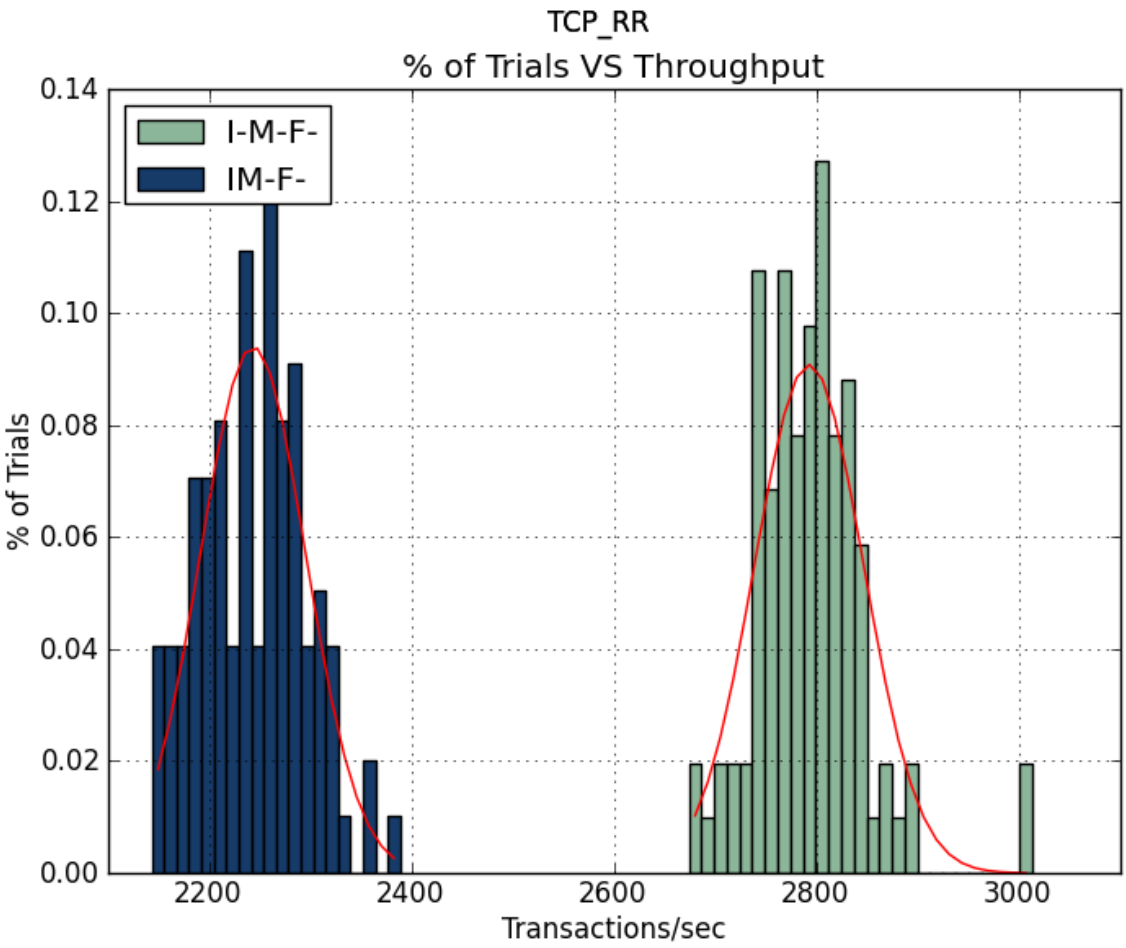

Figure D.14: Histogram comparison of the performance of IPSec using TCP_RR Tests

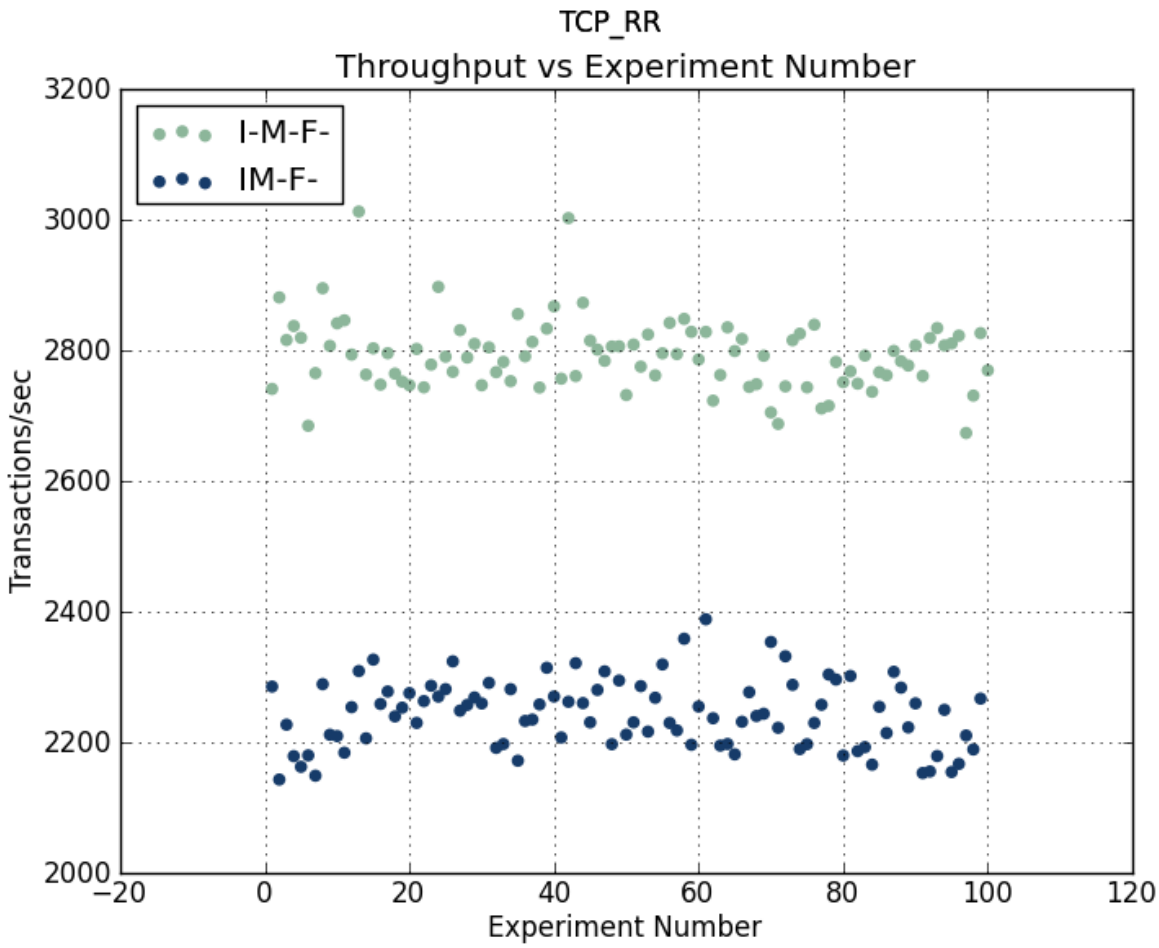

Figure D.15: Scatterplot comparison of the performance of IPSec using TCP_RR Tests

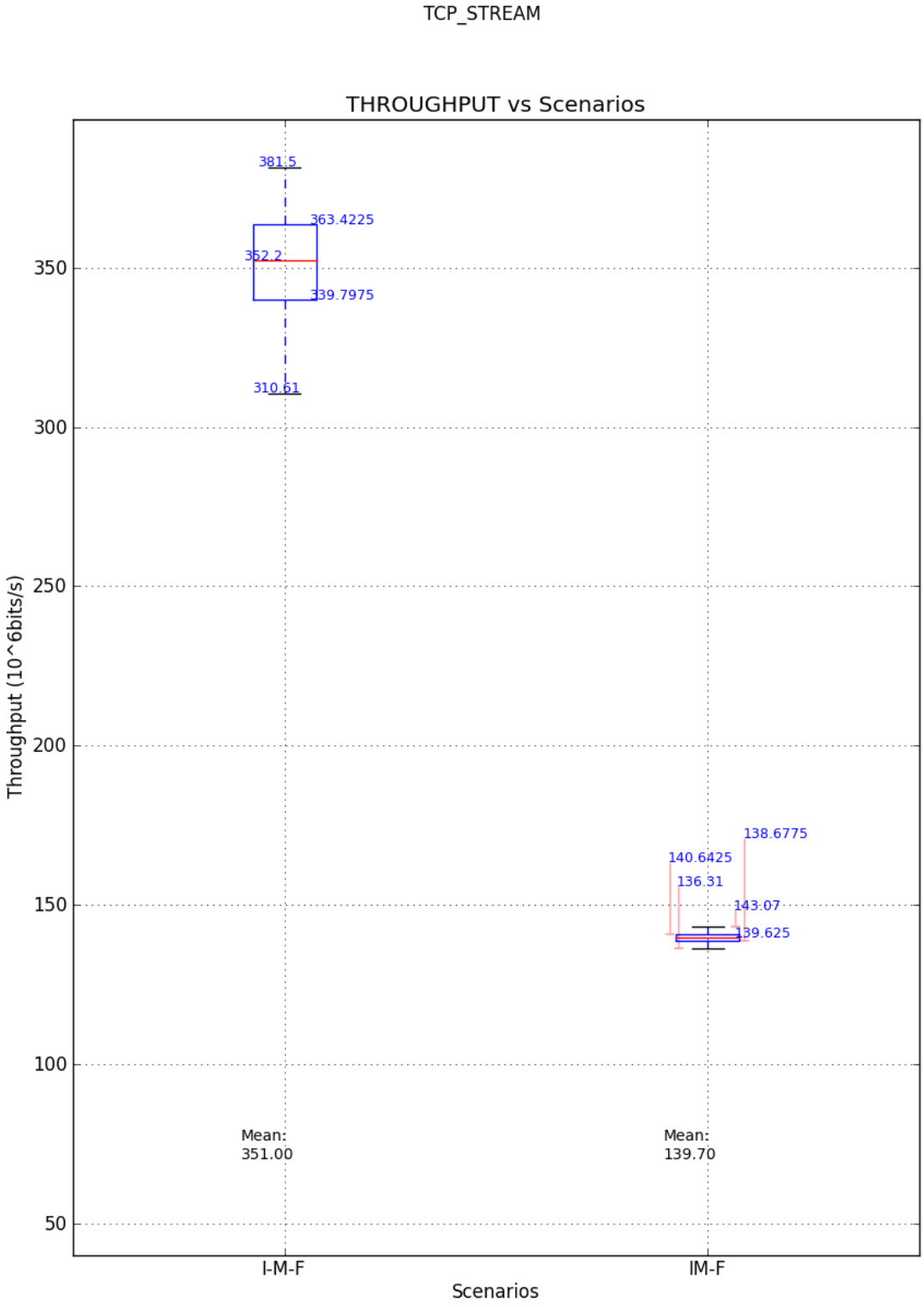

Figure D.16: Boxplots comparison of IPSec using TCP_STREAM Tests

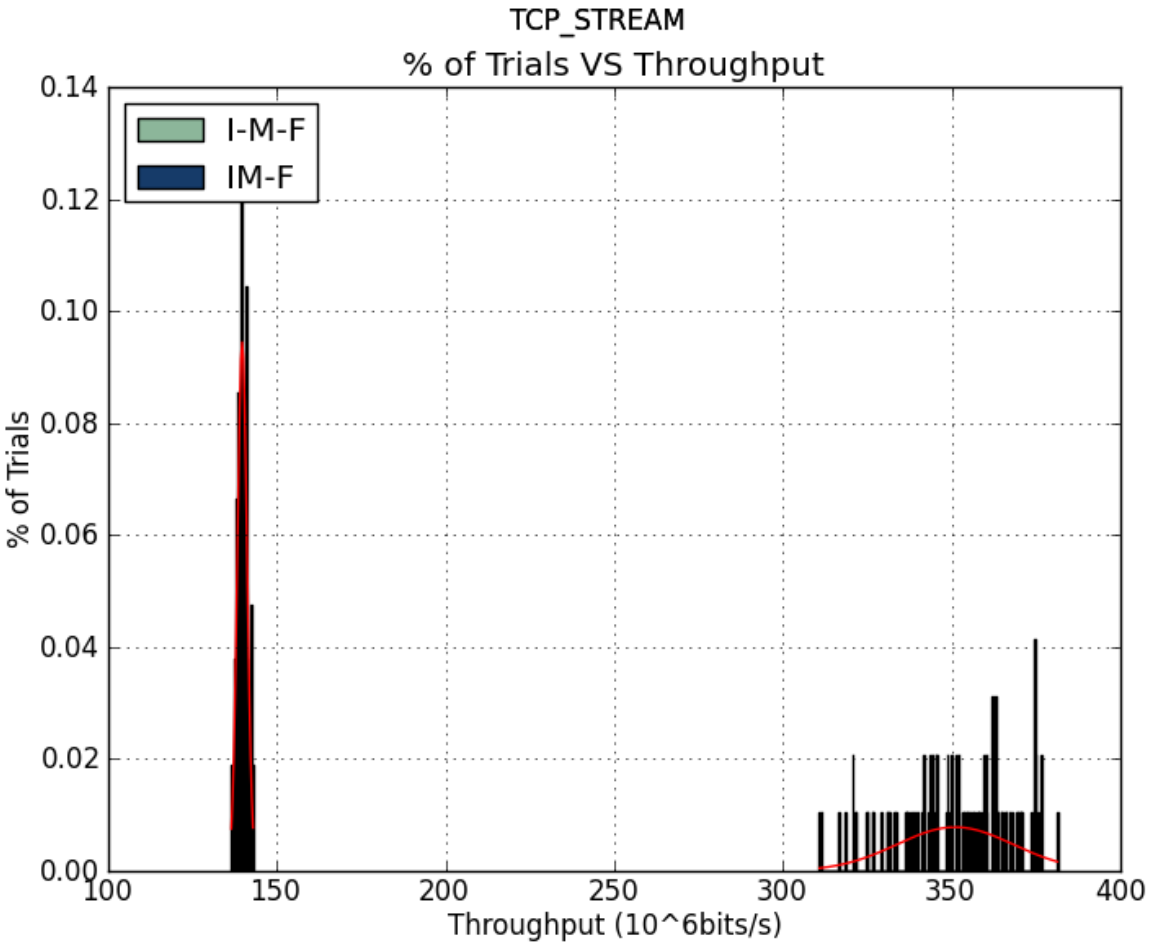

Figure D.17: Histogram comparison of IPSec using TCP_STREAM Tests

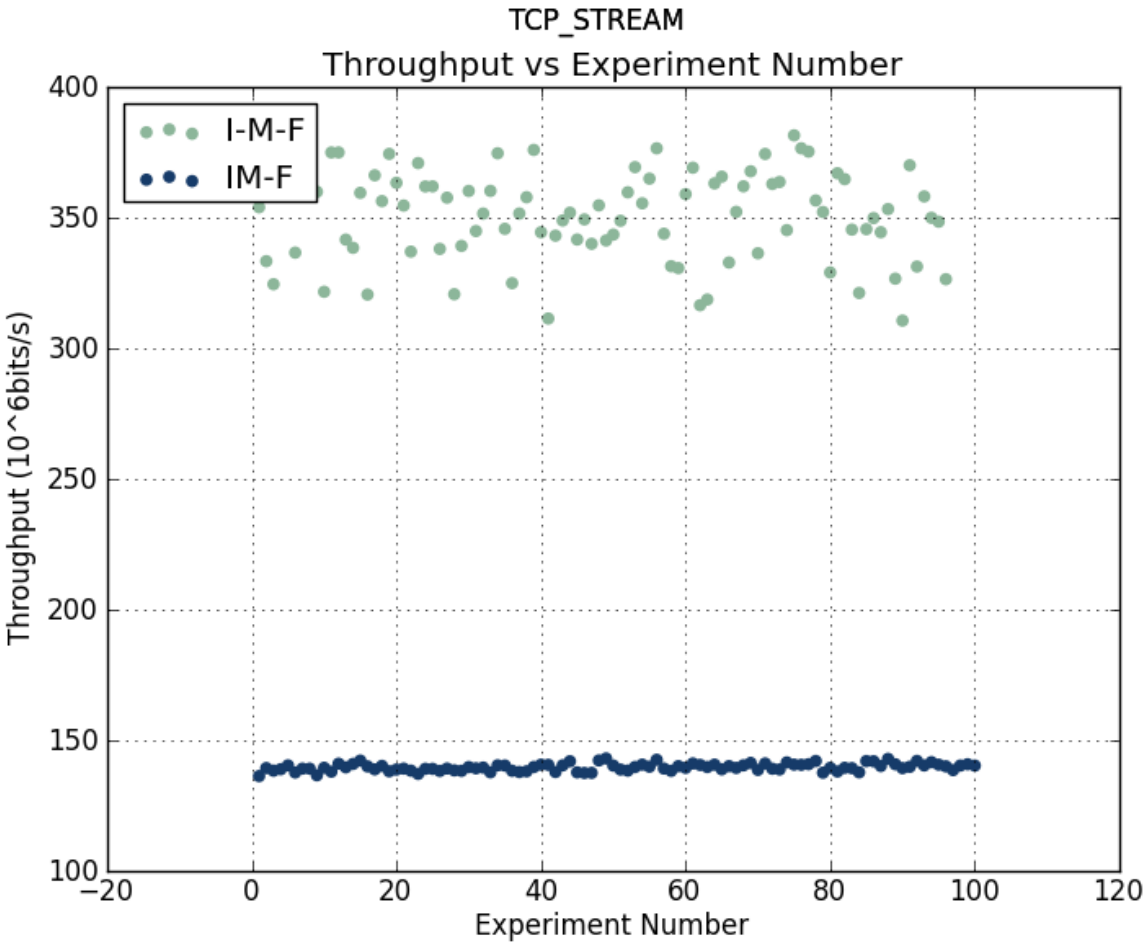

Figure D.18: Scatterplot comparison of IPSec using TCP_STREAM Tests

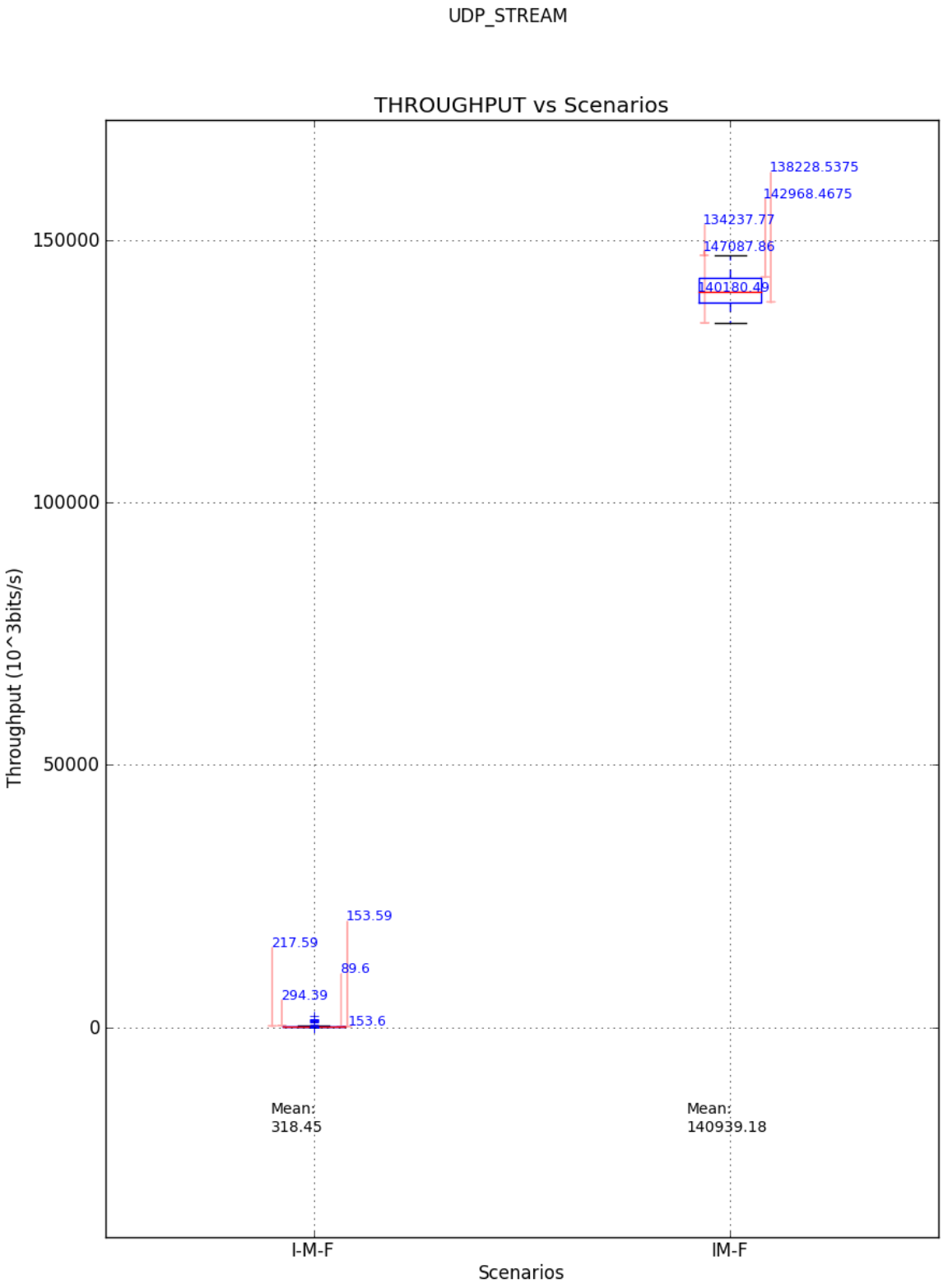

Figure D.19: Boxplots comparison of the performance of IPSec using UDP_STREAM Tests

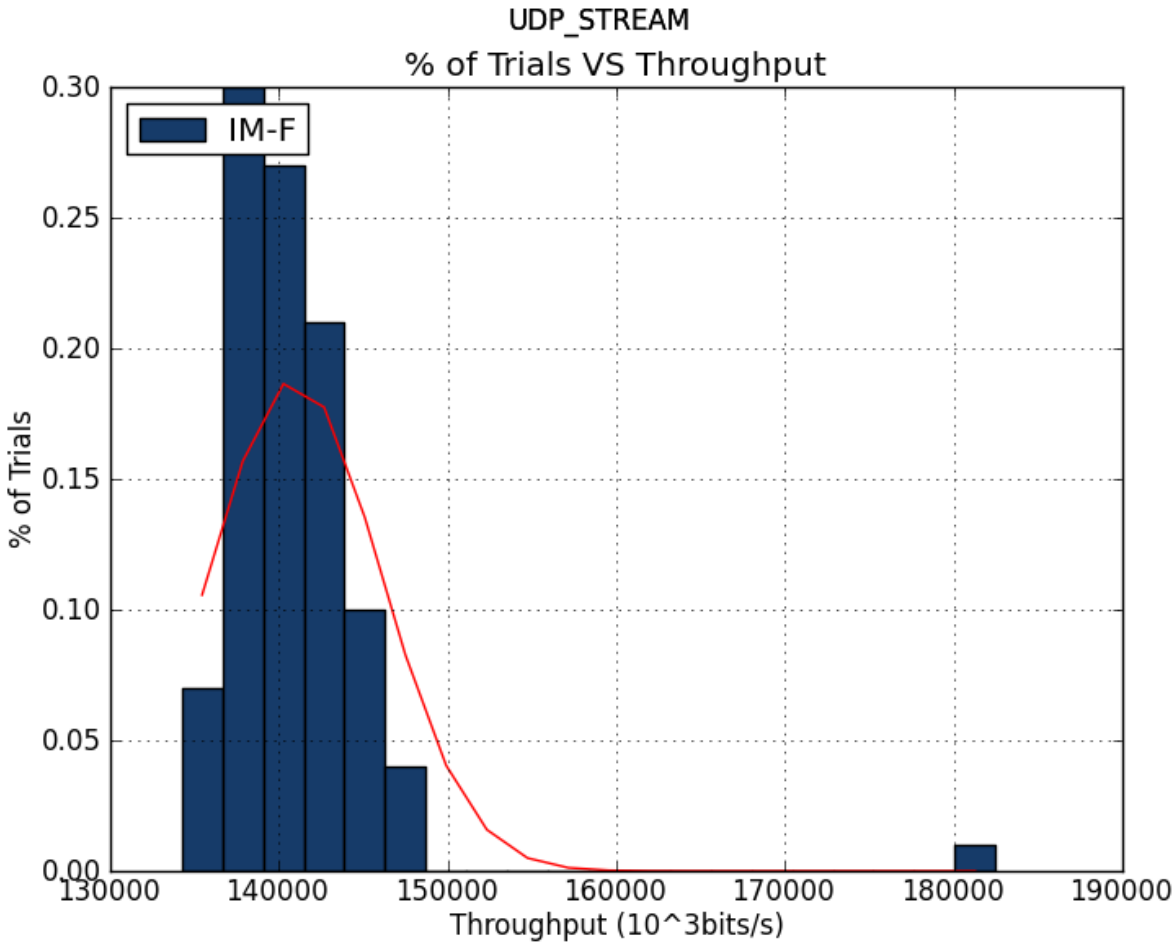

Figure D.20: Histogram comparison of the performance of IPSec using UDP_STREAM Tests

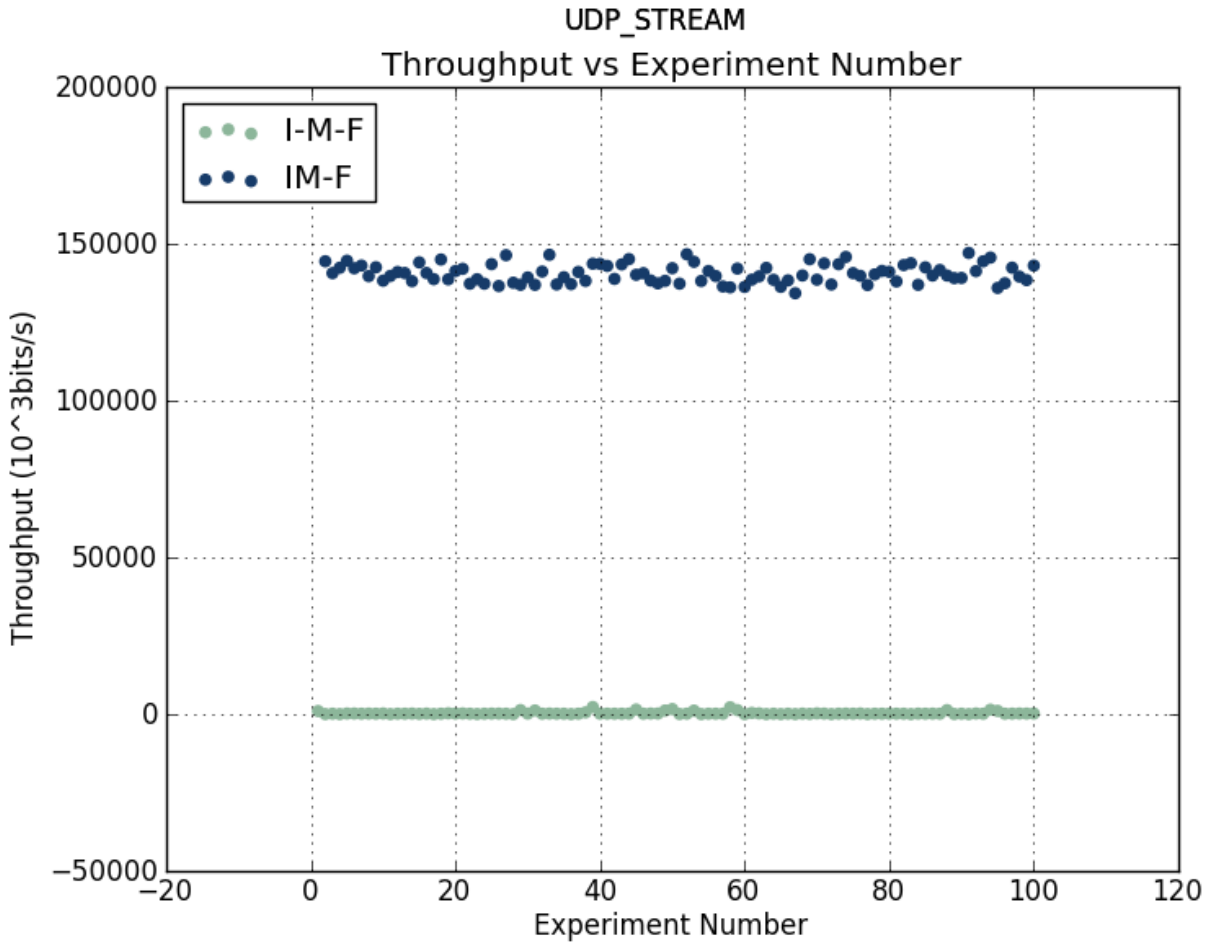

Figure D.21: Scatterplot comparison of the performance of IPSec using UDP_STREAM Tests

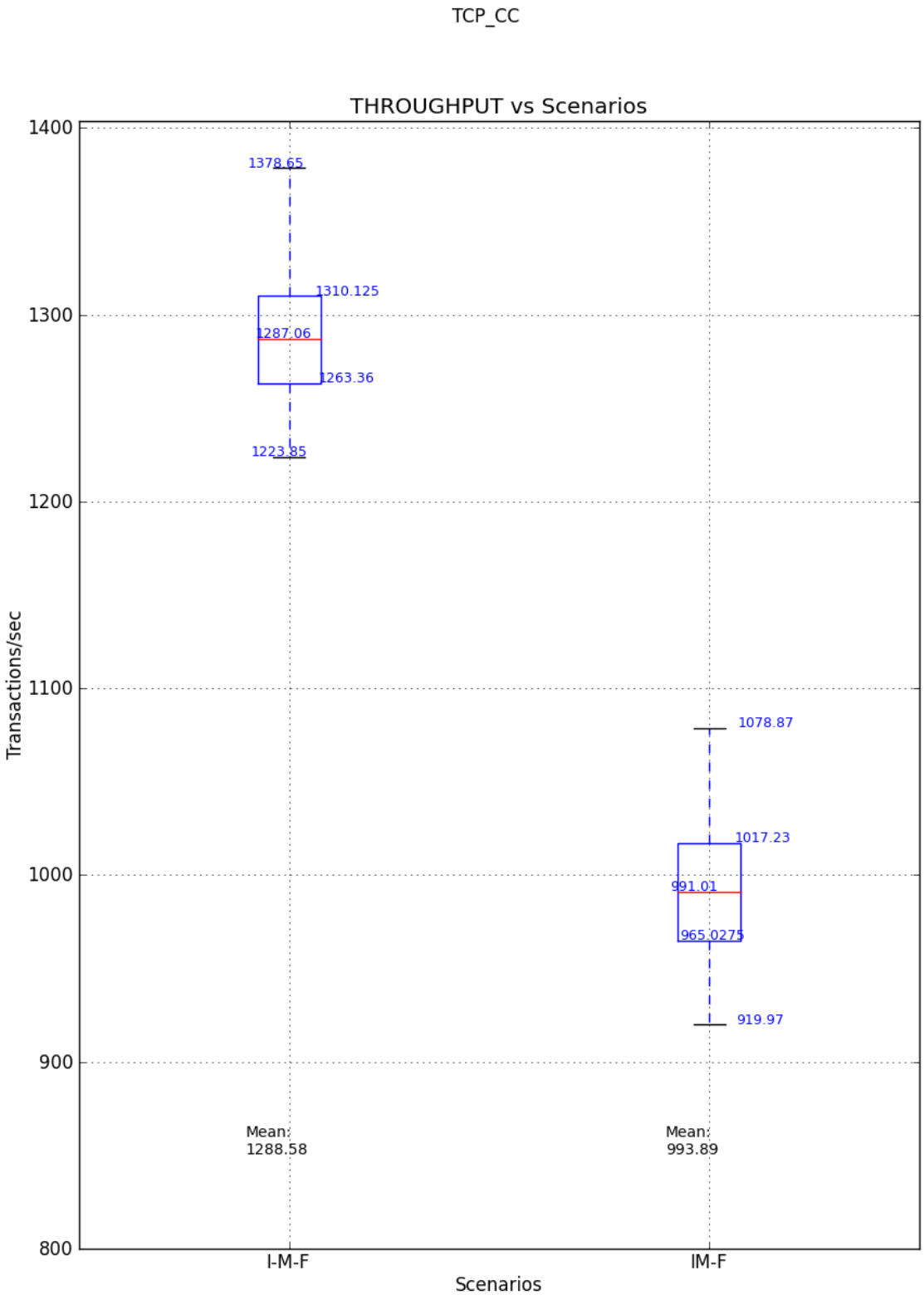

Figure D.22: Boxplots comparison of the performance of IPSec using TCP_CC Tests

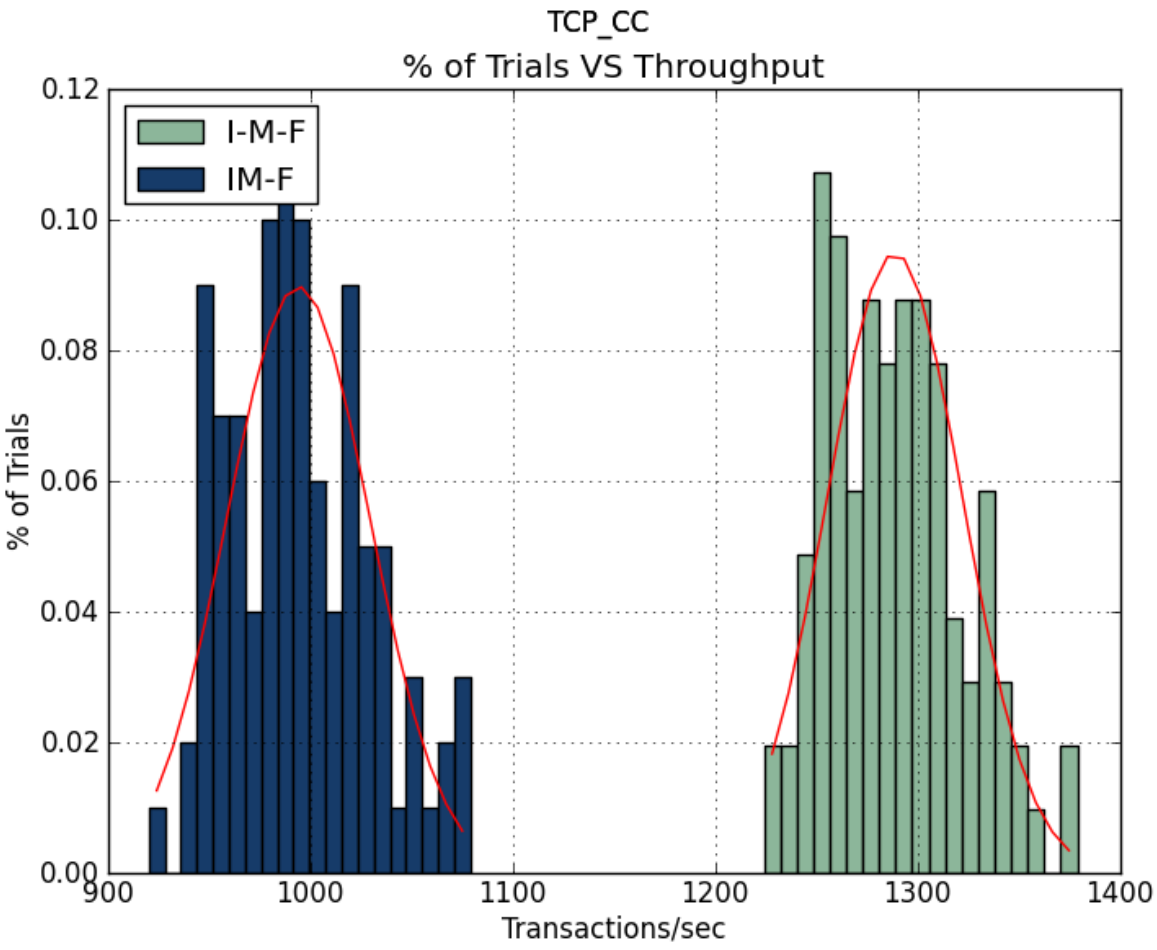

Figure D.23: Histogram comparison of the performance of IPSec using TCP_CC Tests

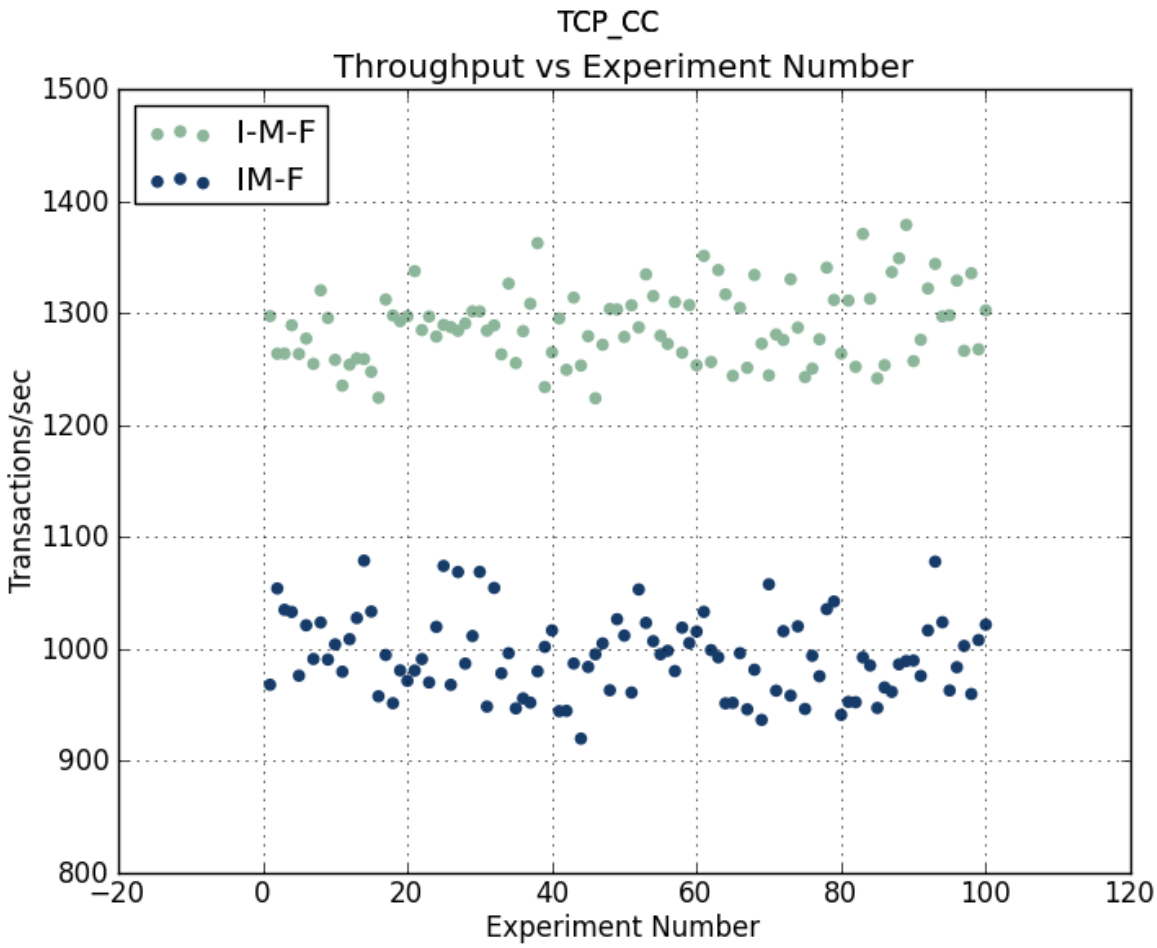

Figure D.24: Scatterplot comparison of the performance of IPSec using TCP_CC Tests

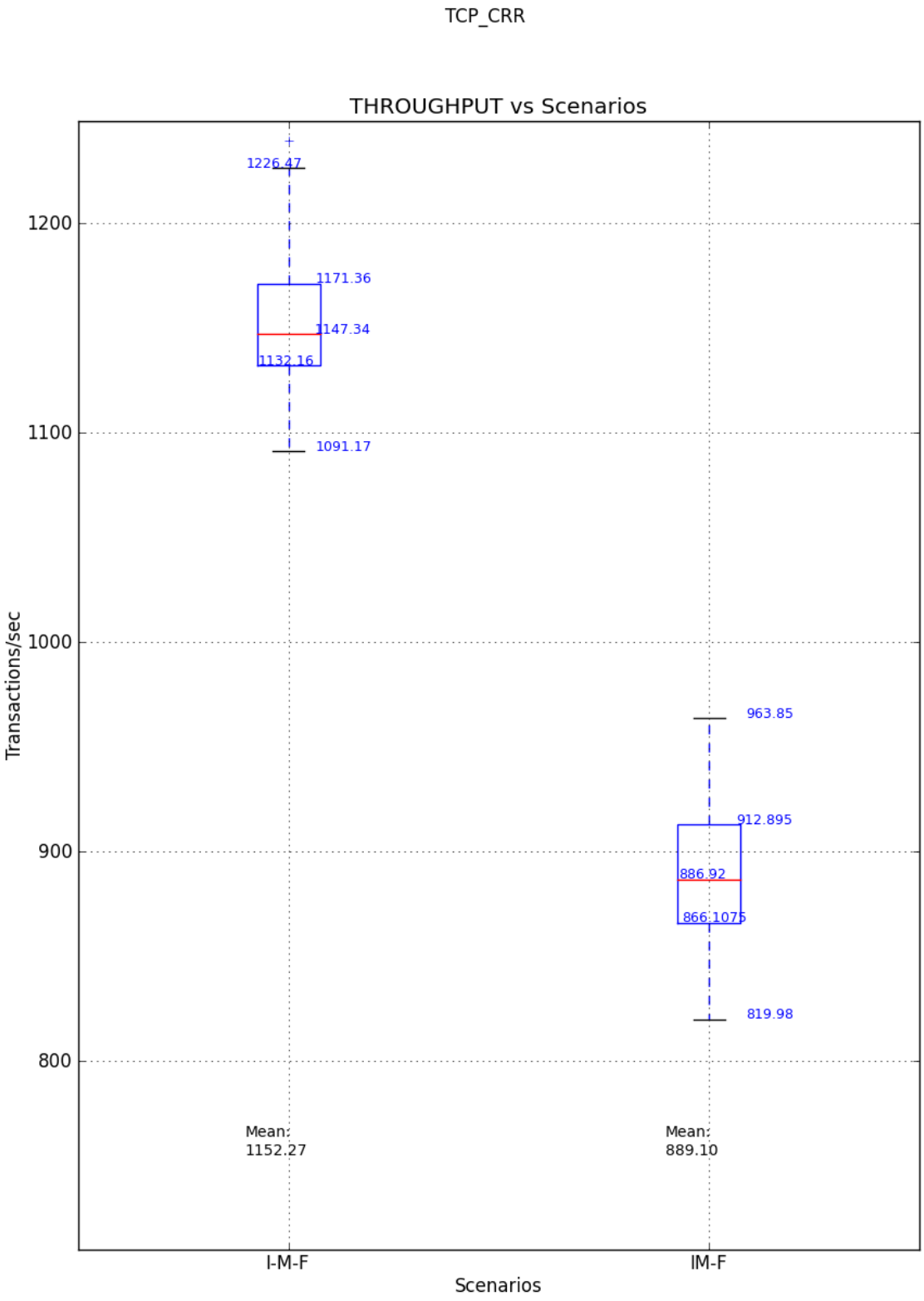

Figure D.25: Boxplots comparison of the performance of IPSec using TCP_CRR Tests

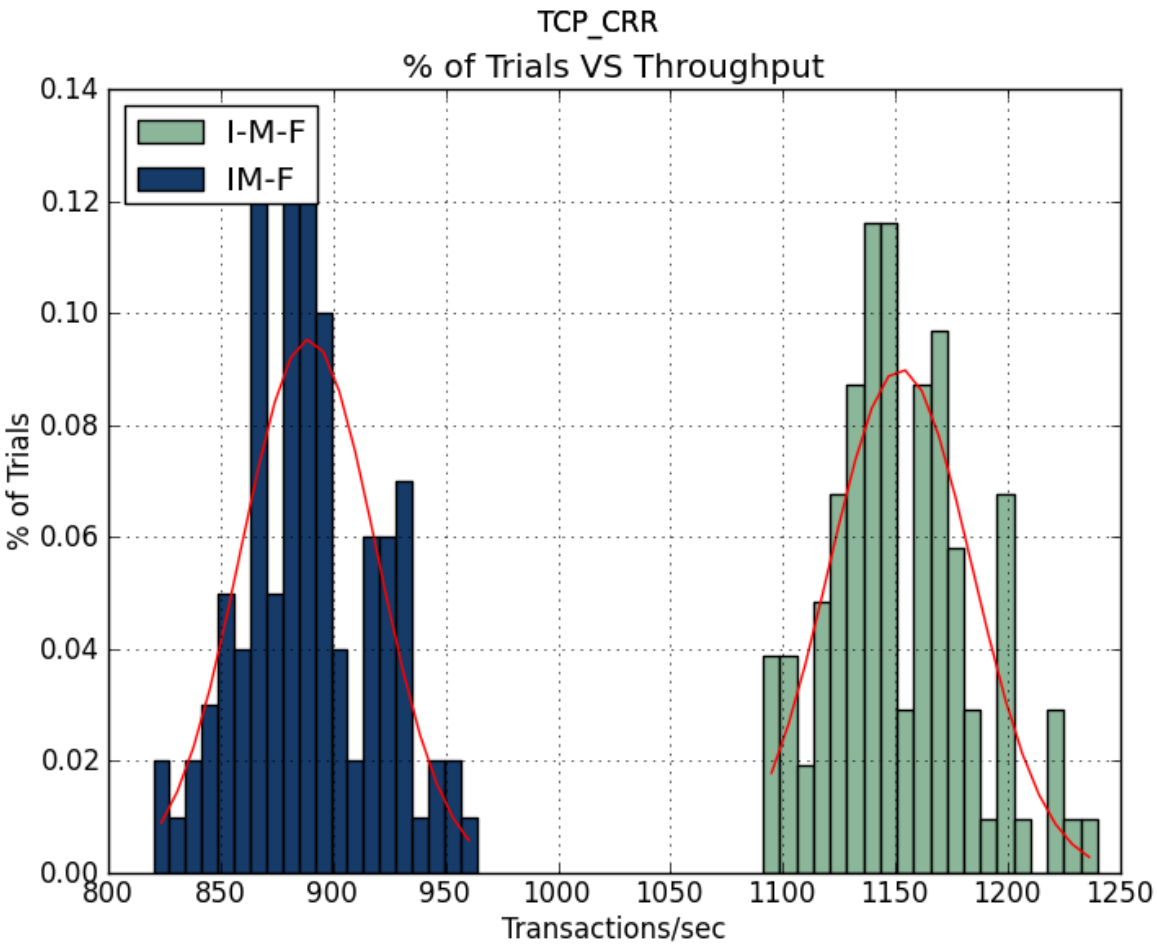

Figure D.26: Histogram comparison of the performance of IPSec using TCP_CRR Tests

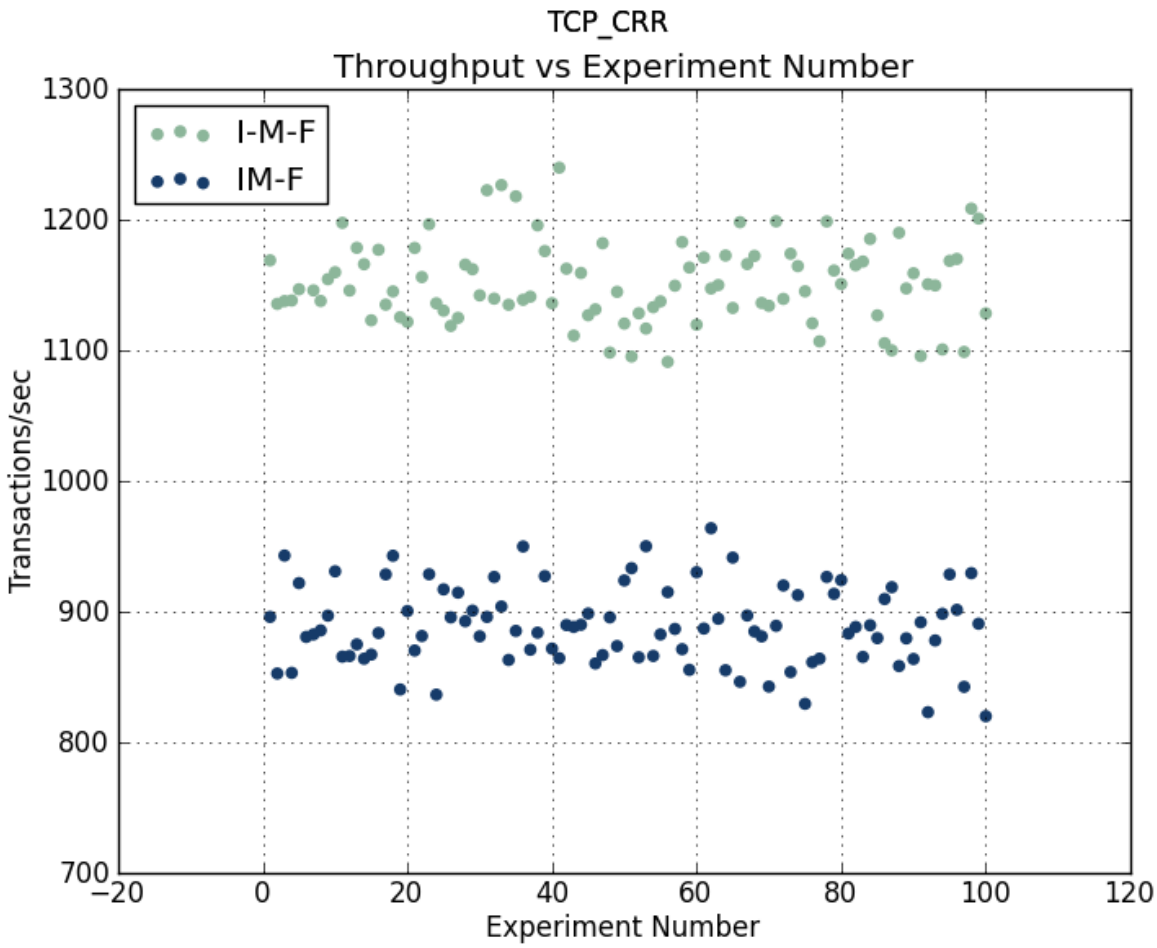

Figure D.27: Scatterplot comparison of the performance of IPSec using TCP_CRR Tests

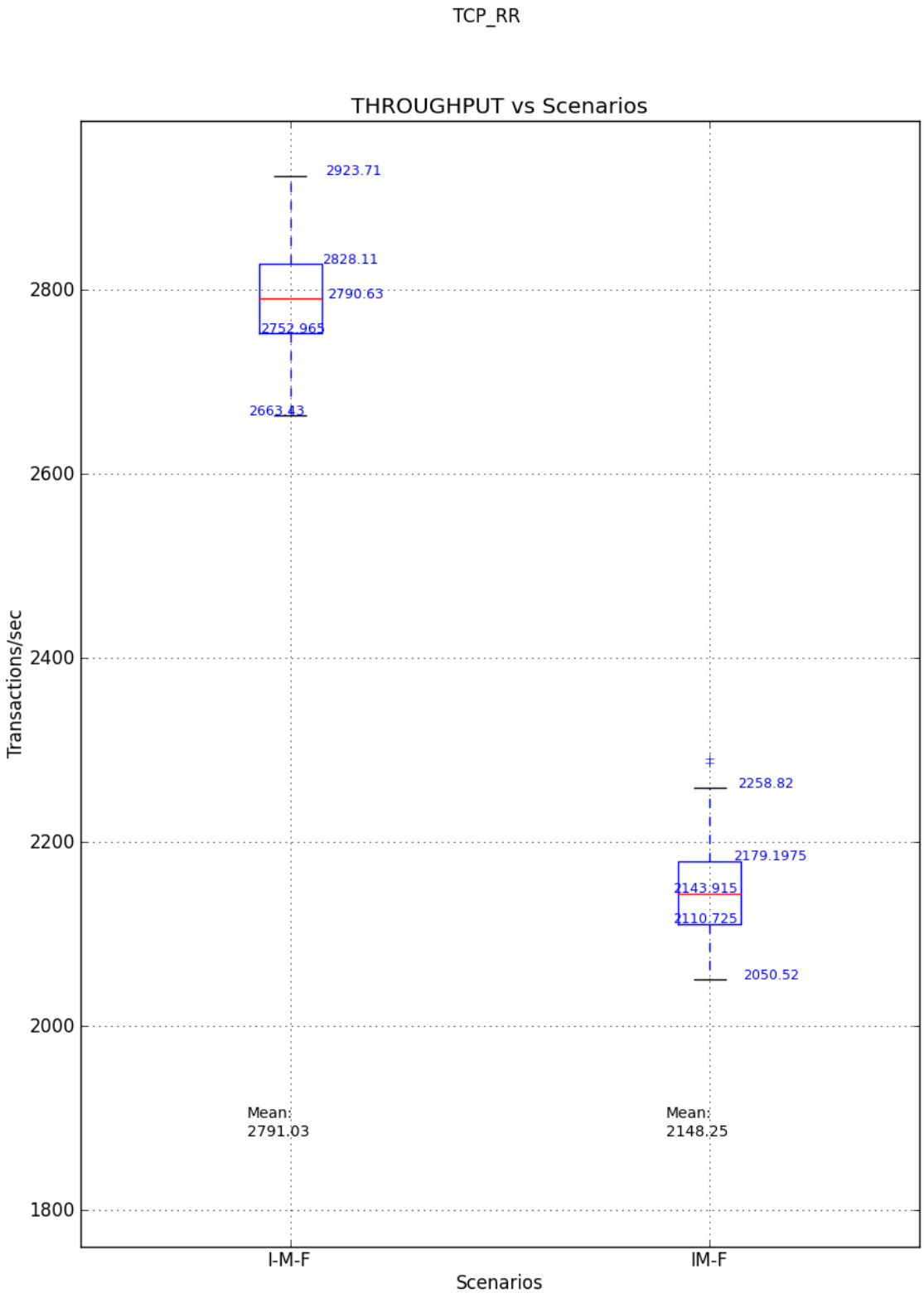

Figure D.28: Boxplots comparison of the performance of IPSec using TCP_RR Tests

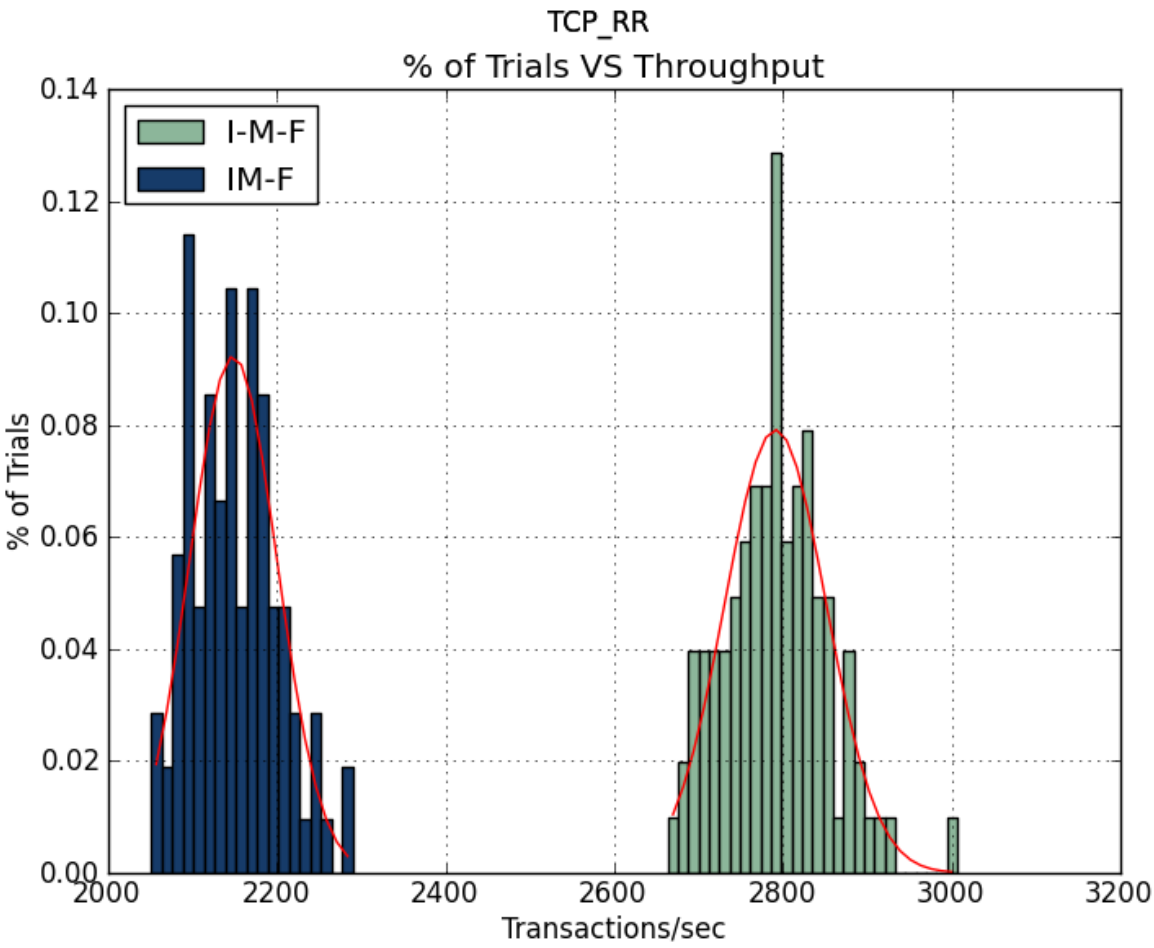

Figure D.29: Histogram comparison of the performance of IPSec using TCP_RR Tests

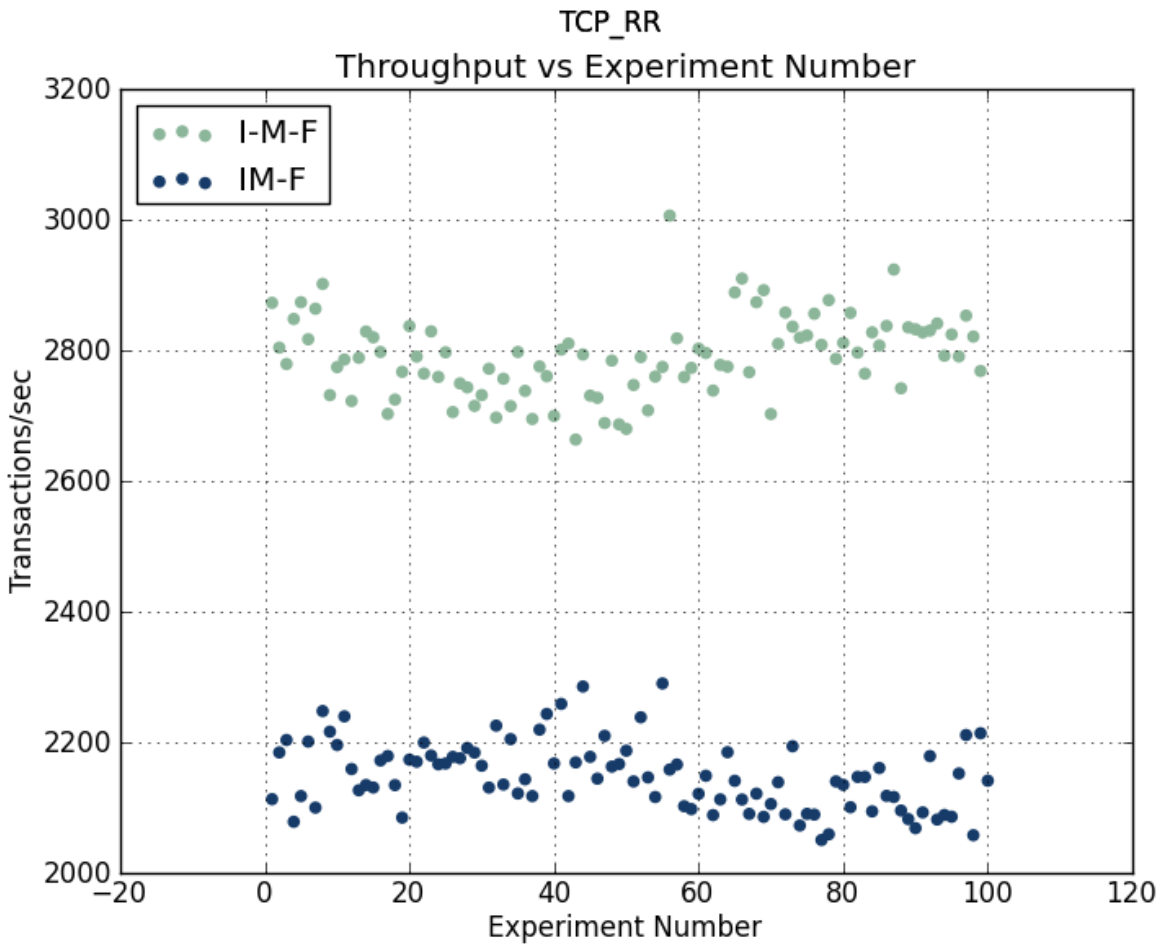

Figure D.30: Scatterplot comparison of the performance of IPSec using TCP_RR Tests

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E:
Results of MYSEA Comparison

This appendix contains the results of the benchmark tests used for the analysis of the performance overhead of the MYSEA proxy.

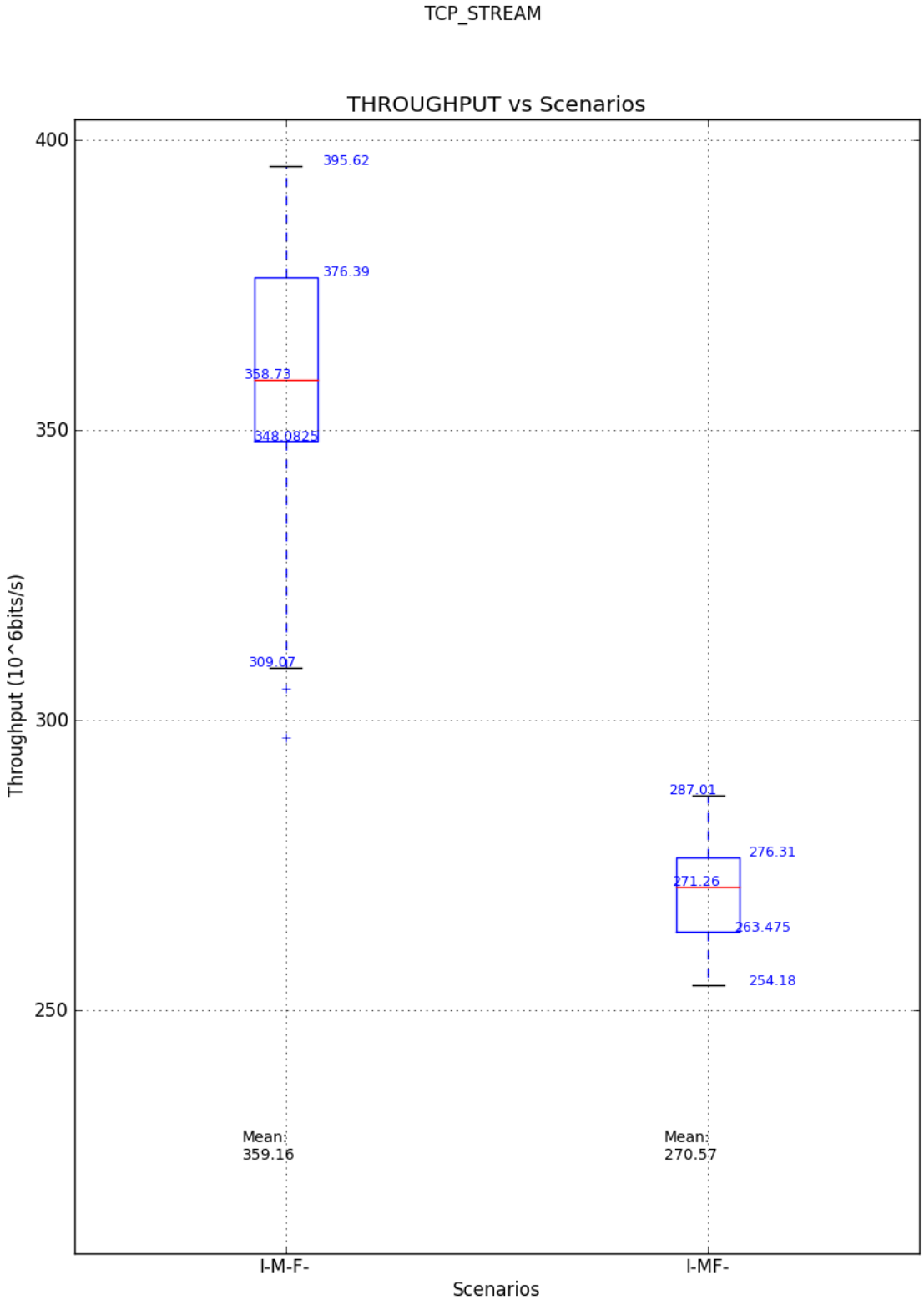

Figure E.1: Boxplots comparison of MYSEA processes using TCP_STREAM Tests

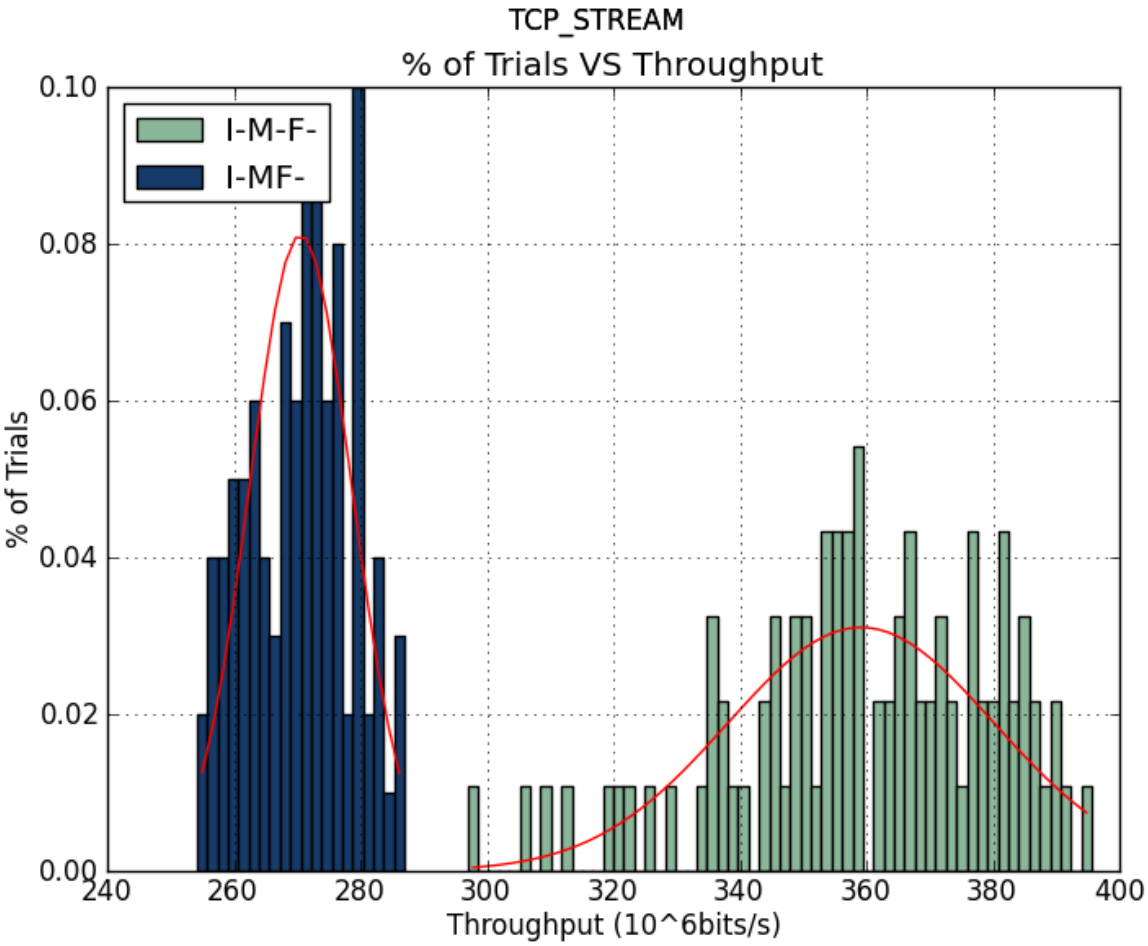

Figure E.2: Histogram comparison of MYSEA processes using TCP_STREAM Tests

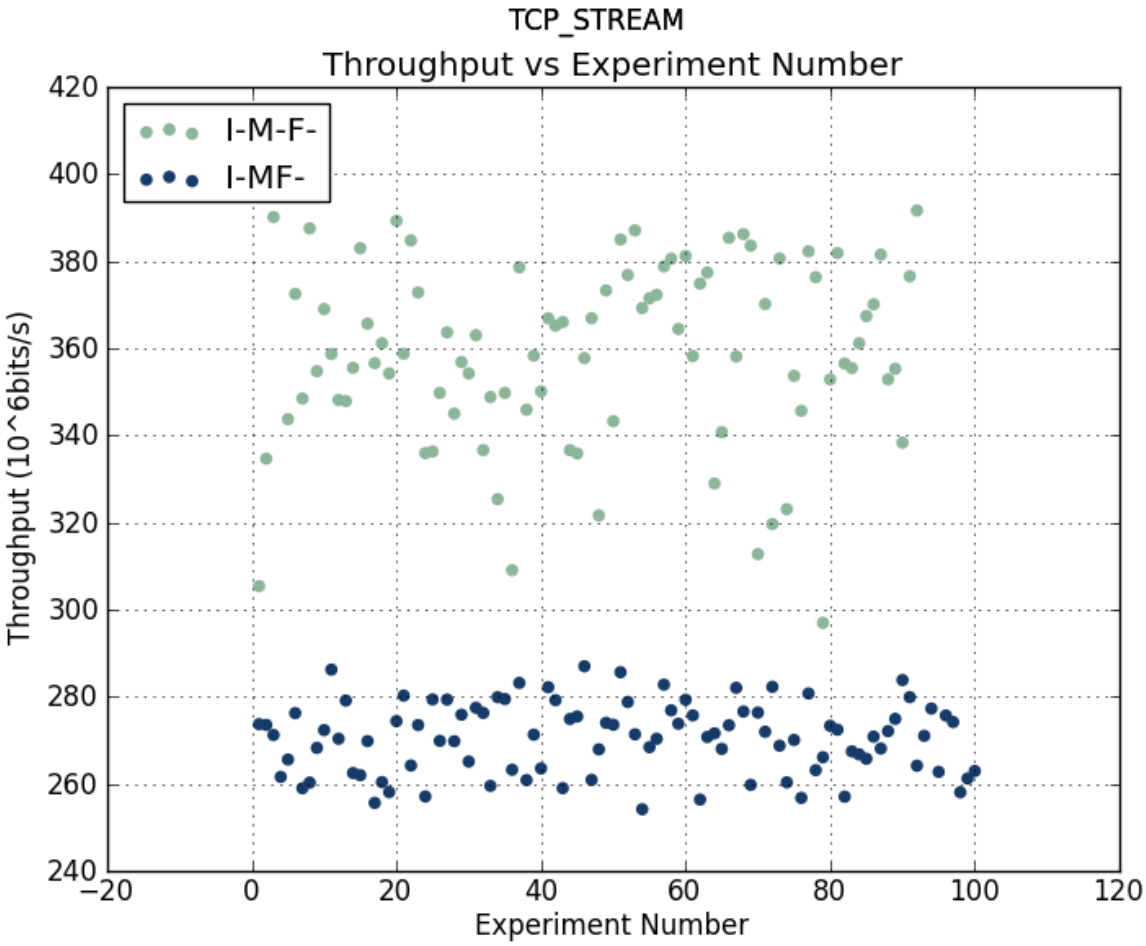

Figure E.3: Scatterplot comparison of MYSEA processes using TCP_STREAM Tests

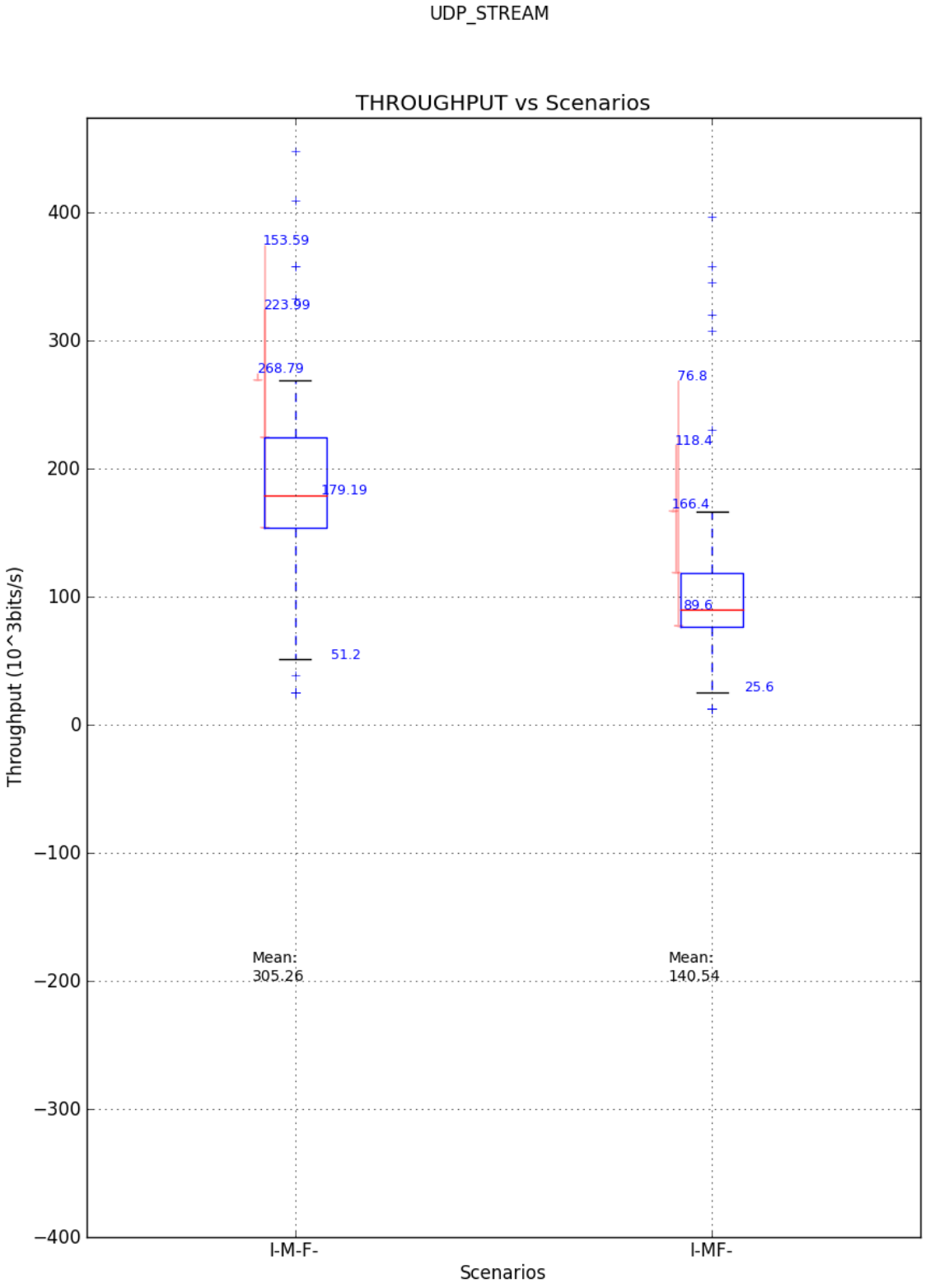

Figure E.4: Boxplots comparison of the performance of IPSec using UDP_STREAM Tests

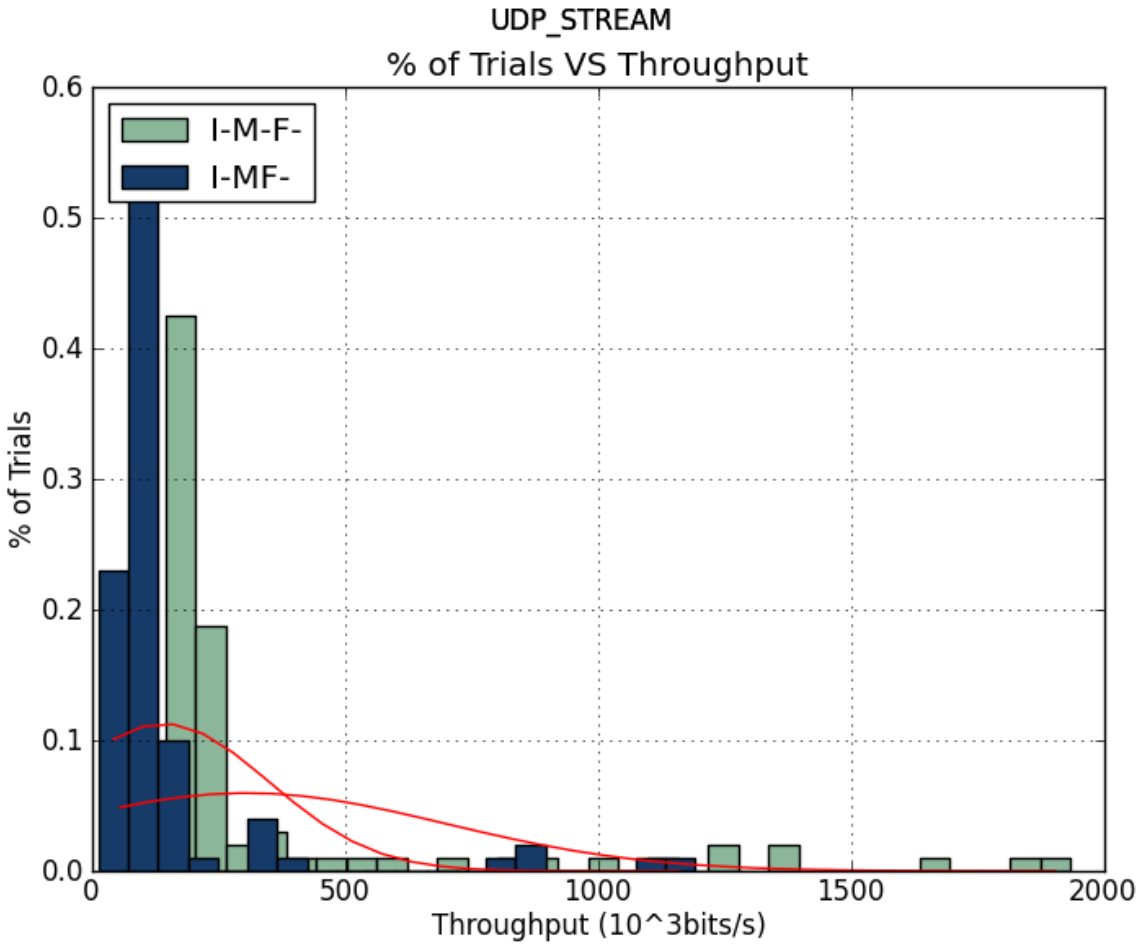

Figure E.5: Histogram comparison of the performance of IPSec using UDP_STREAM Tests

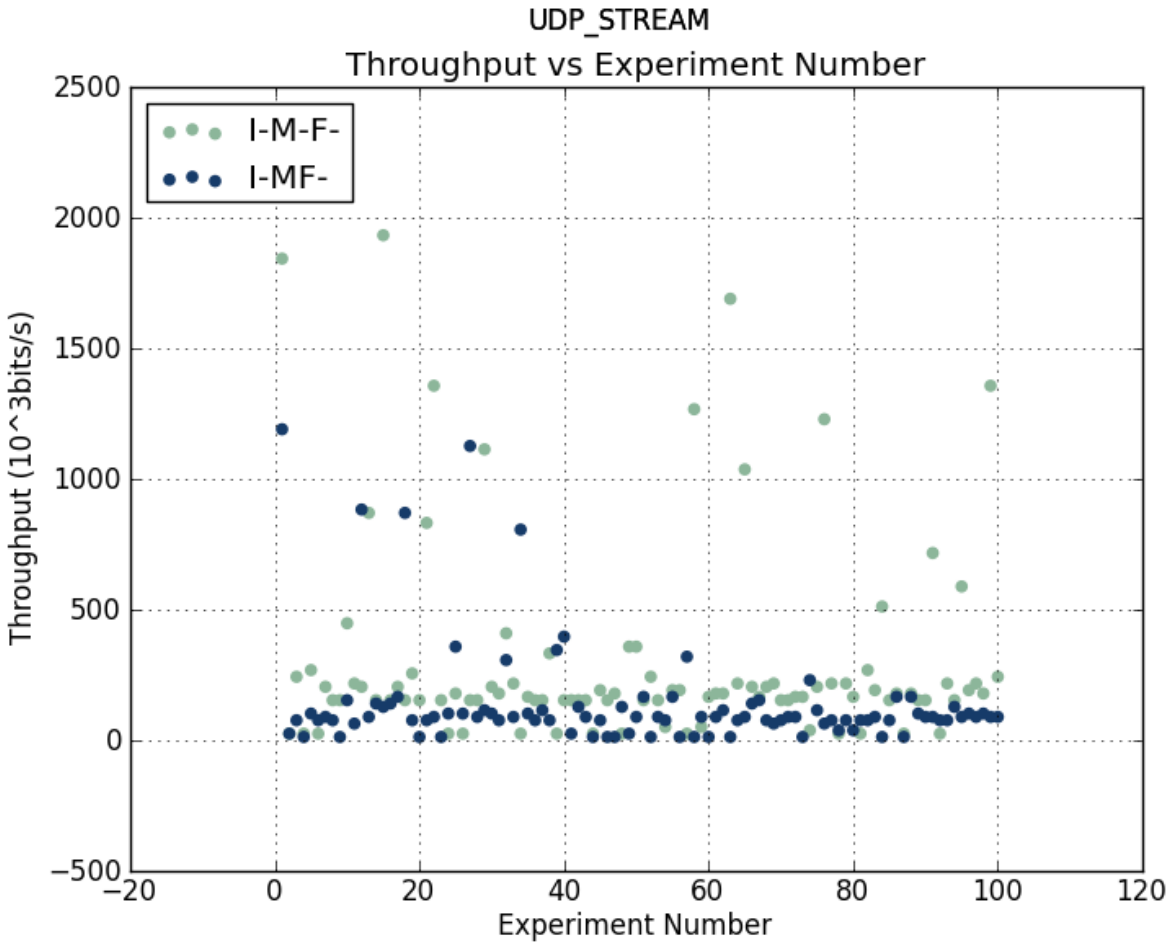

Figure E.6: Scatterplot comparison of the performance of IPSec using UDP_STREAM Tests

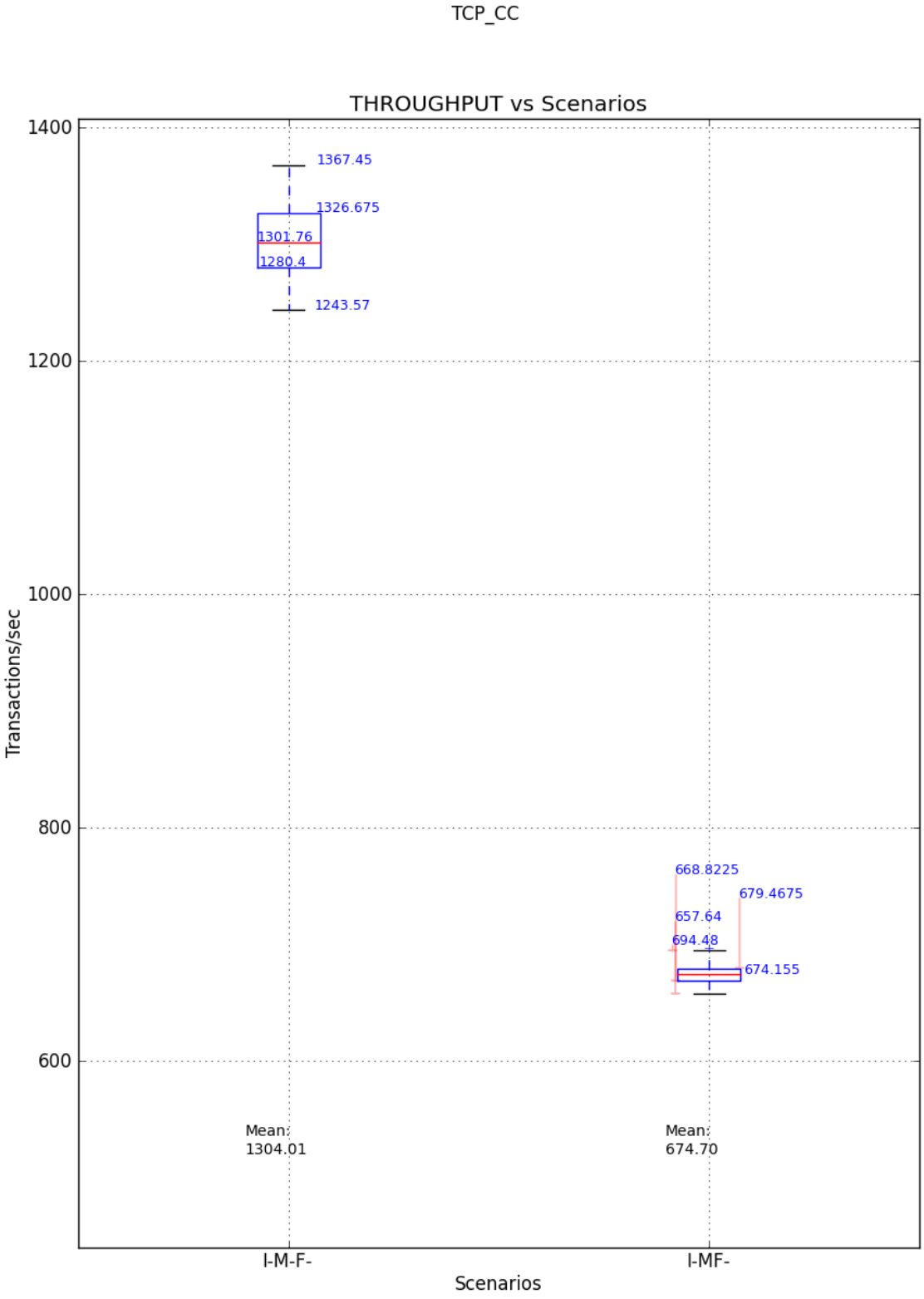

Figure E.7: Boxplots comparison of the performance of IPSec using TCP_CC Tests

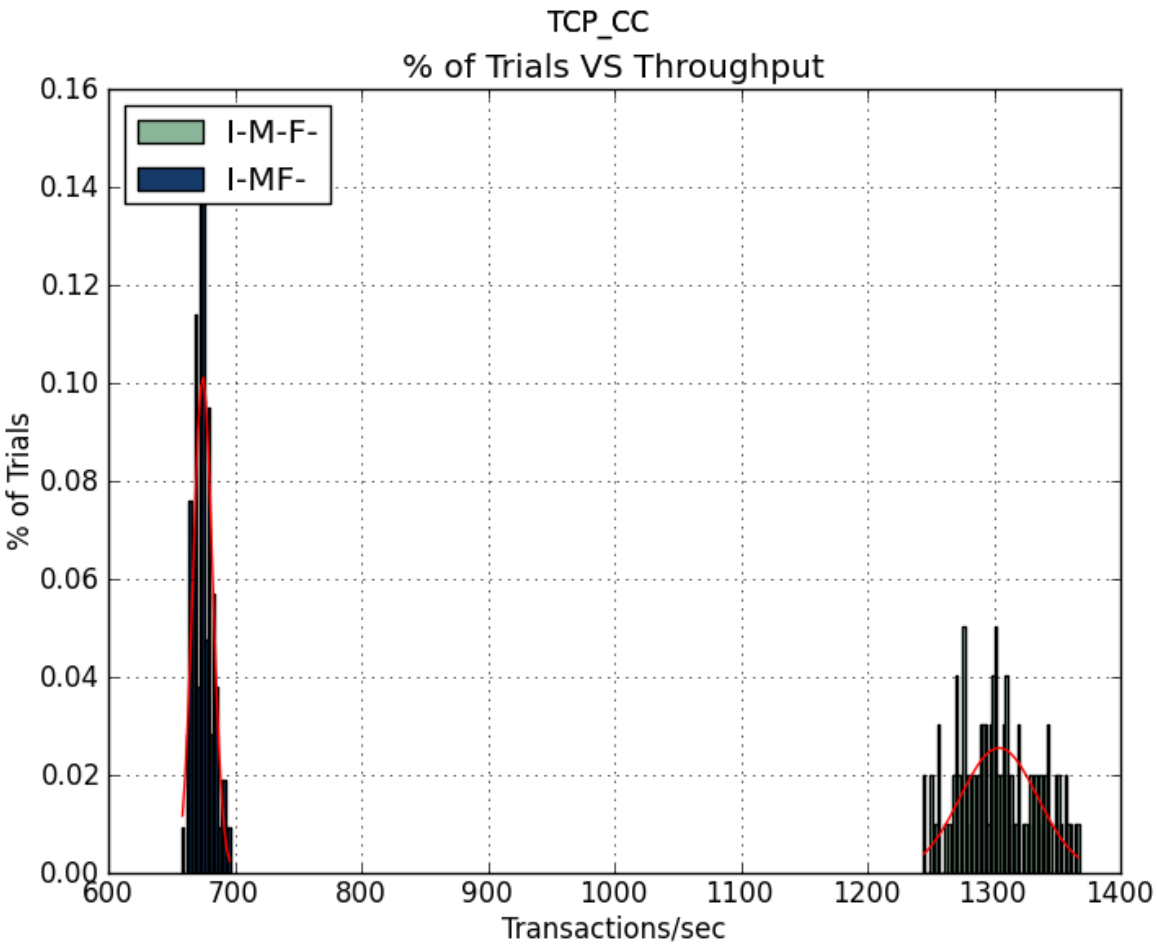

Figure E.8: Histogram comparison of the performance of IPSec using TCP_CC Tests

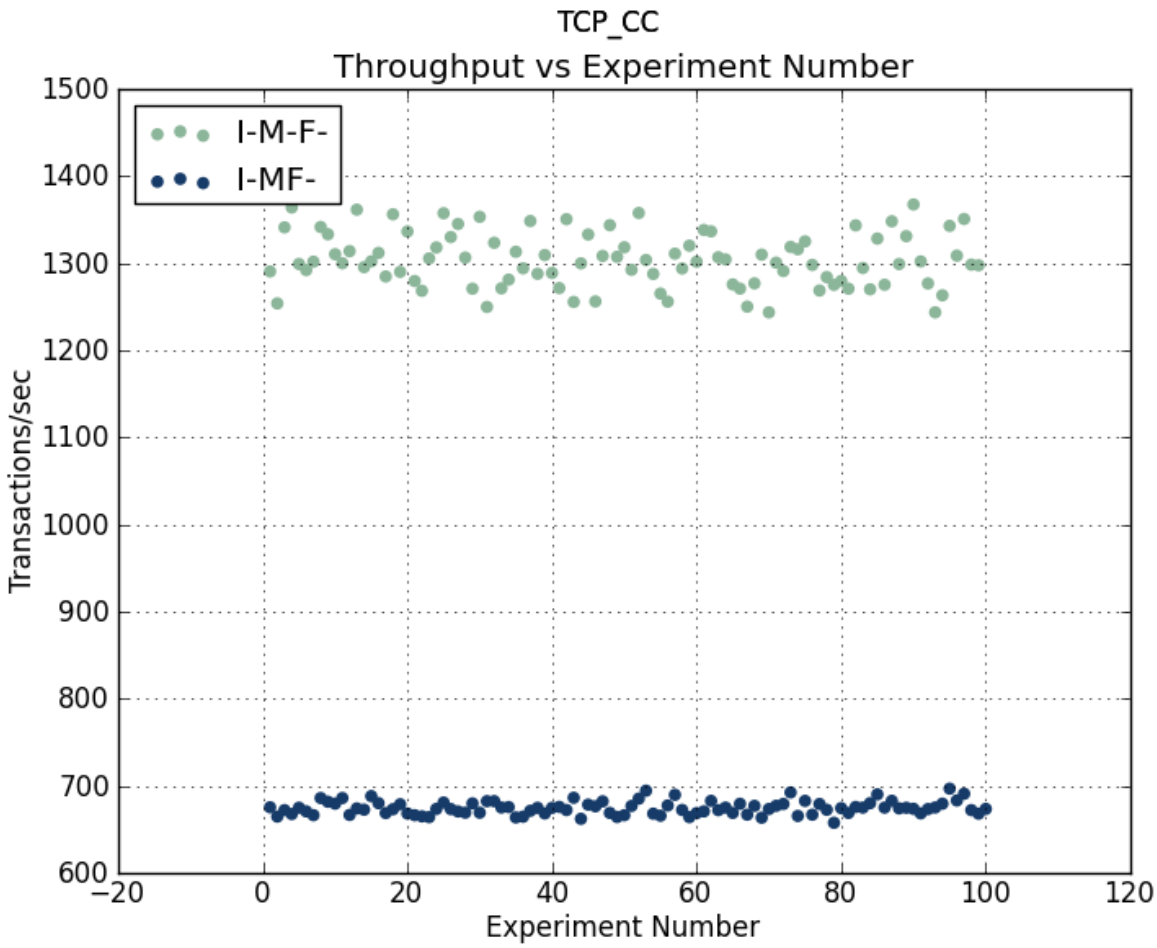

Figure E.9: Scatterplot comparison of the performance of IPSec using TCP_CC Tests

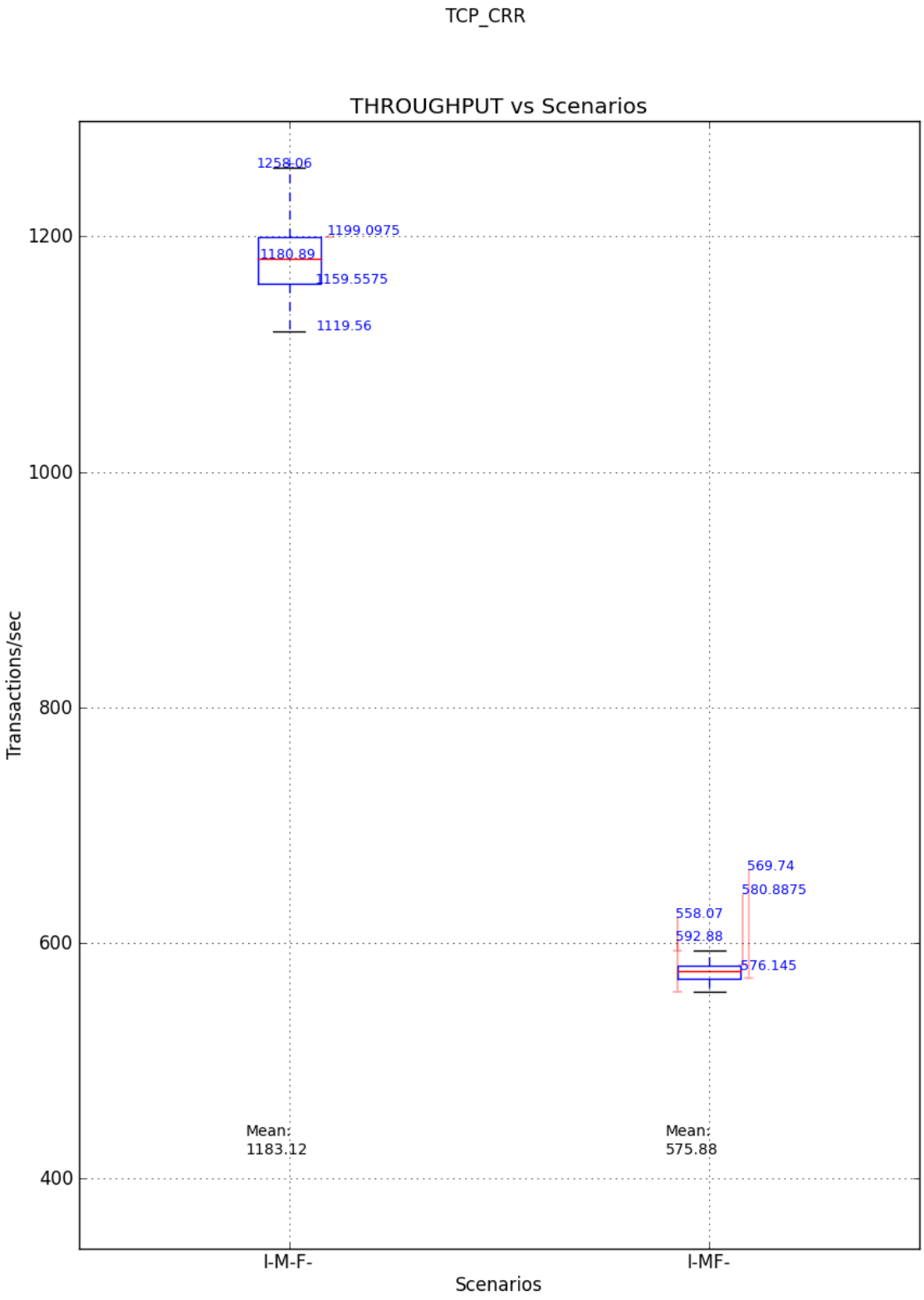

Figure E.10: Boxplots comparison of the performance of IPSec using TCP_CRR Tests

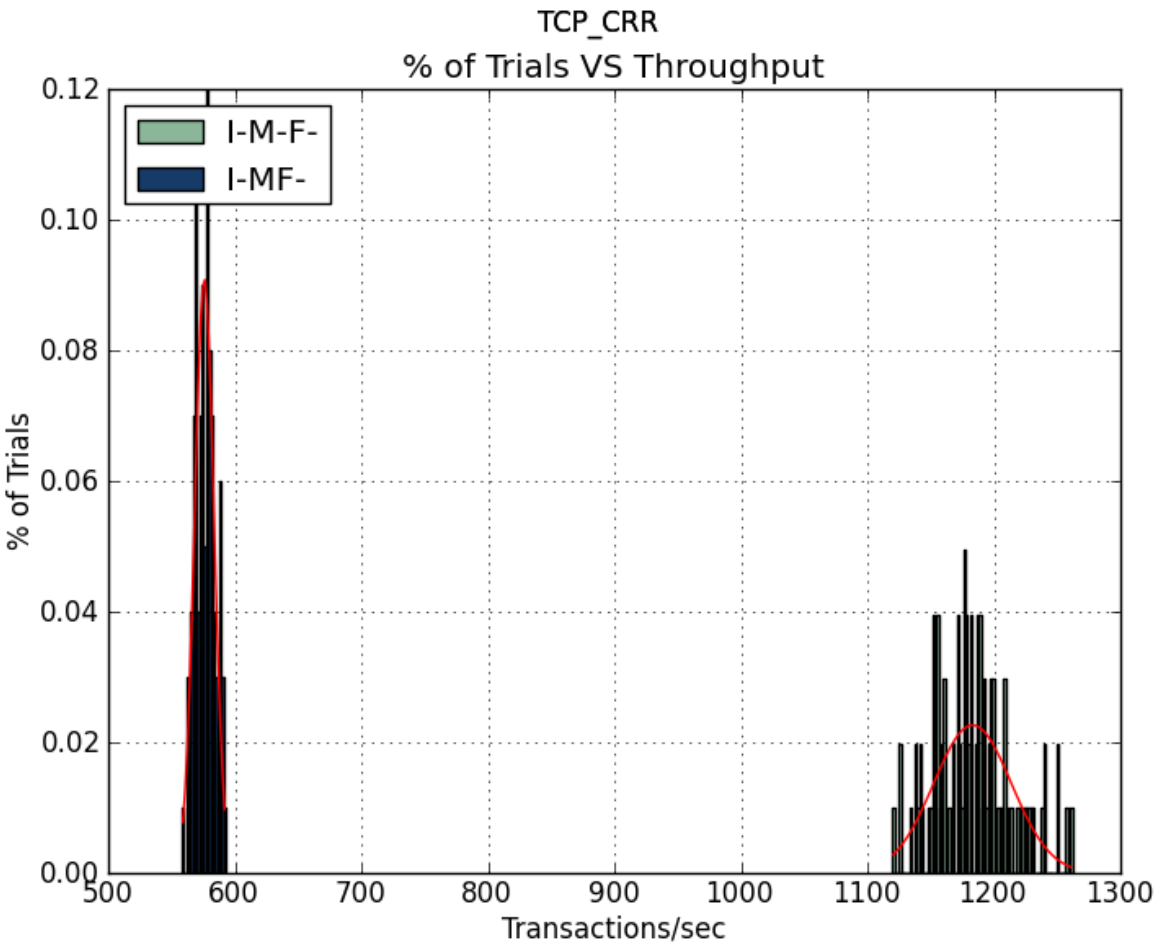

Figure E.11: Histogram comparison of the performance of IPSec using TCP_CRR Tests

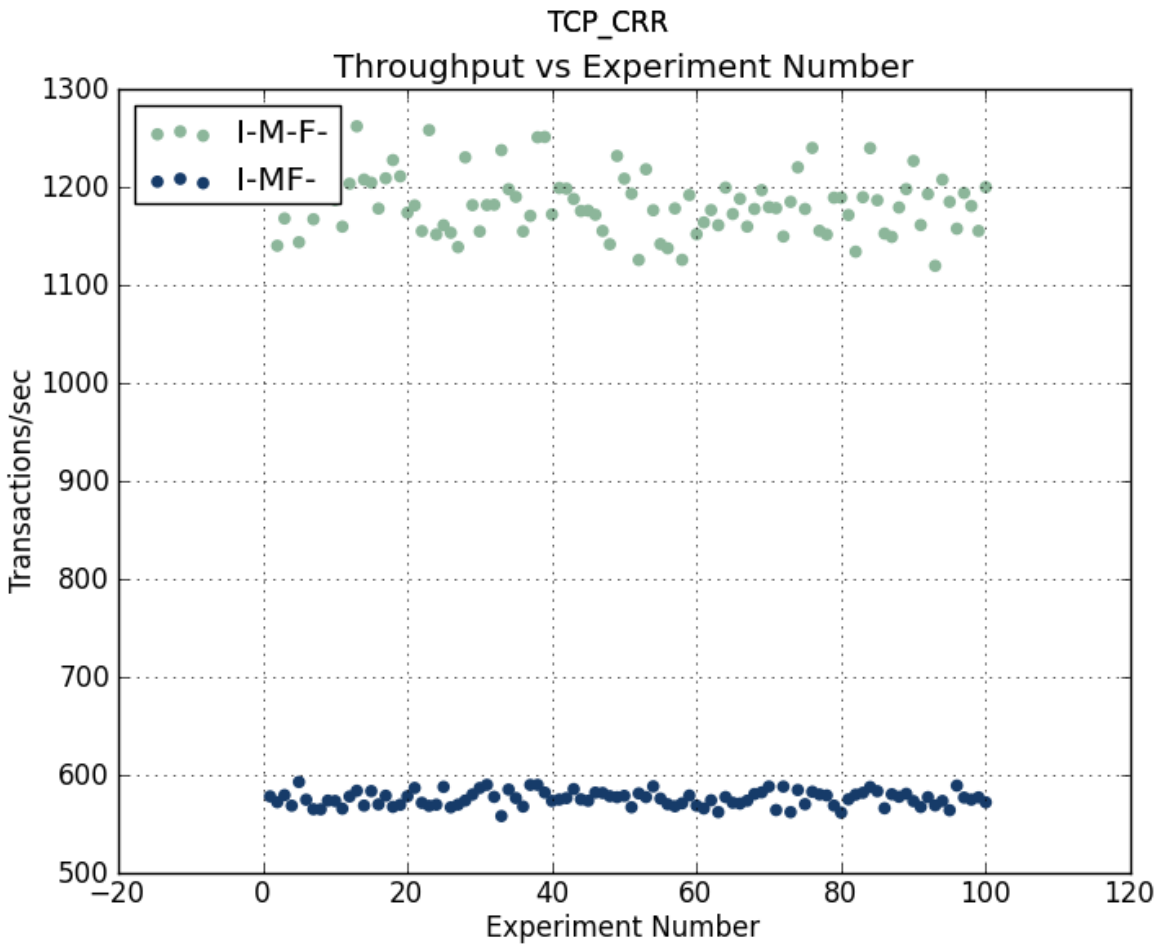

Figure E.12: Scatterplot comparison of the performance of IPSec using TCP_CRR Tests

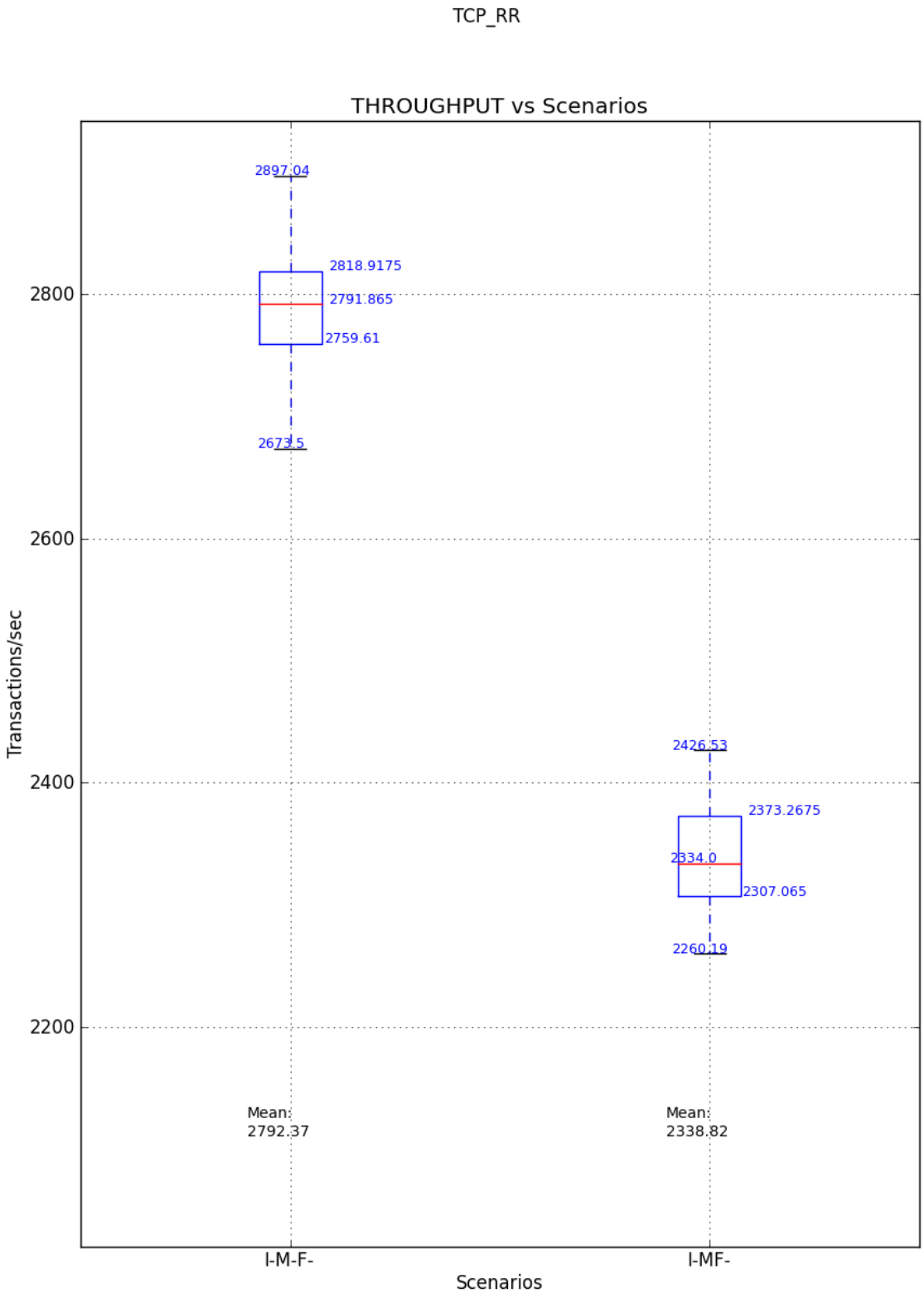

Figure E.13: Boxplots comparison of the performance of IPSec using TCP_RR Tests

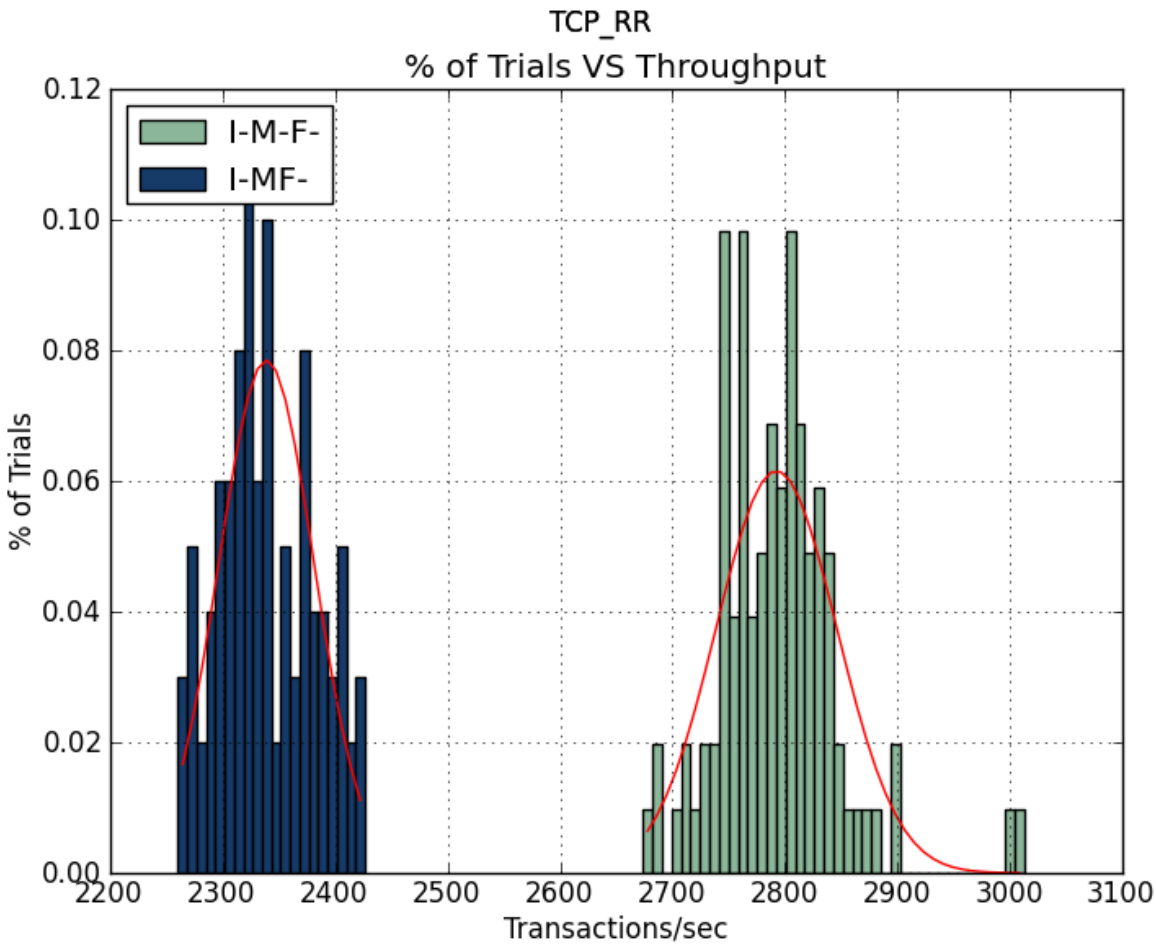

Figure E.14: Histogram comparison of the performance of IPSec using TCP_RR Tests

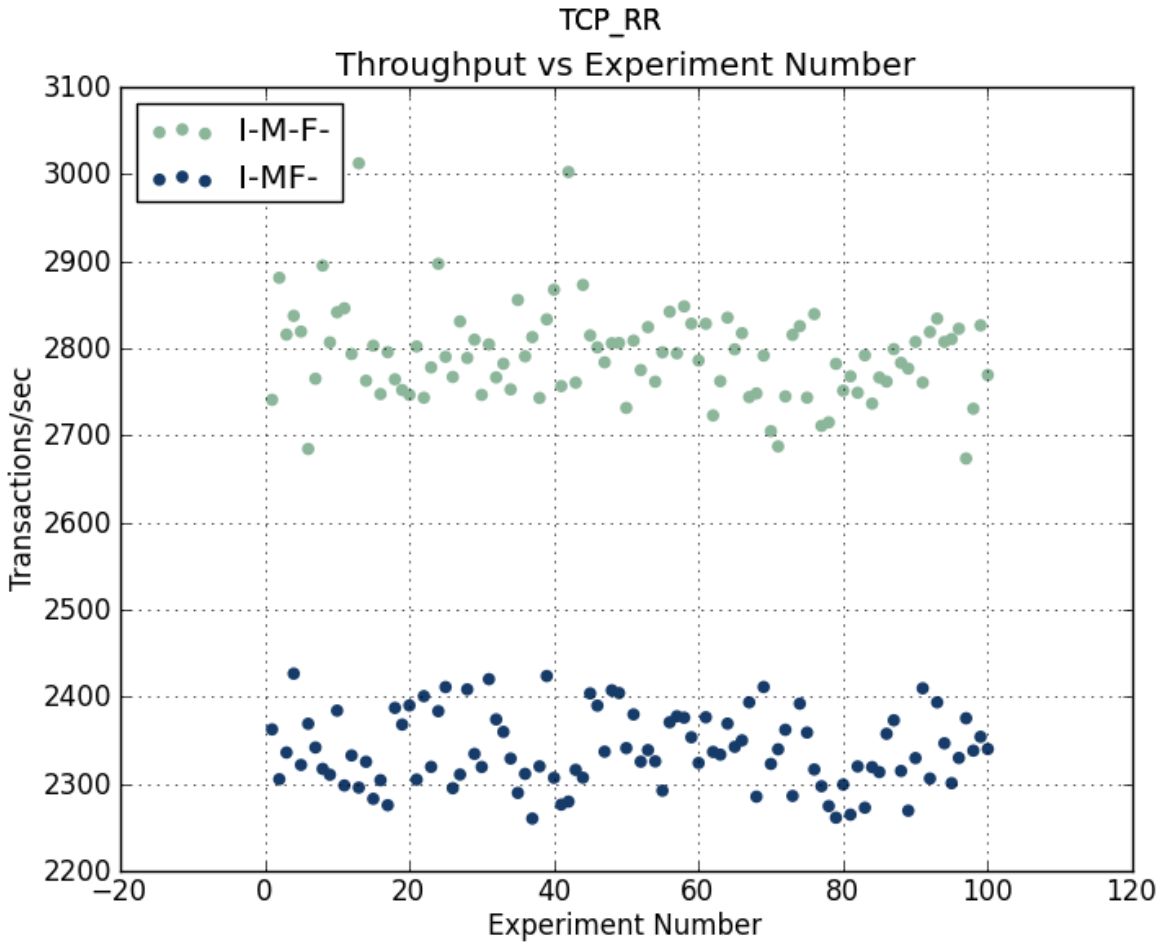

Figure E.15: Scatterplot comparison of the performance of IPSec using TCP_RR Tests

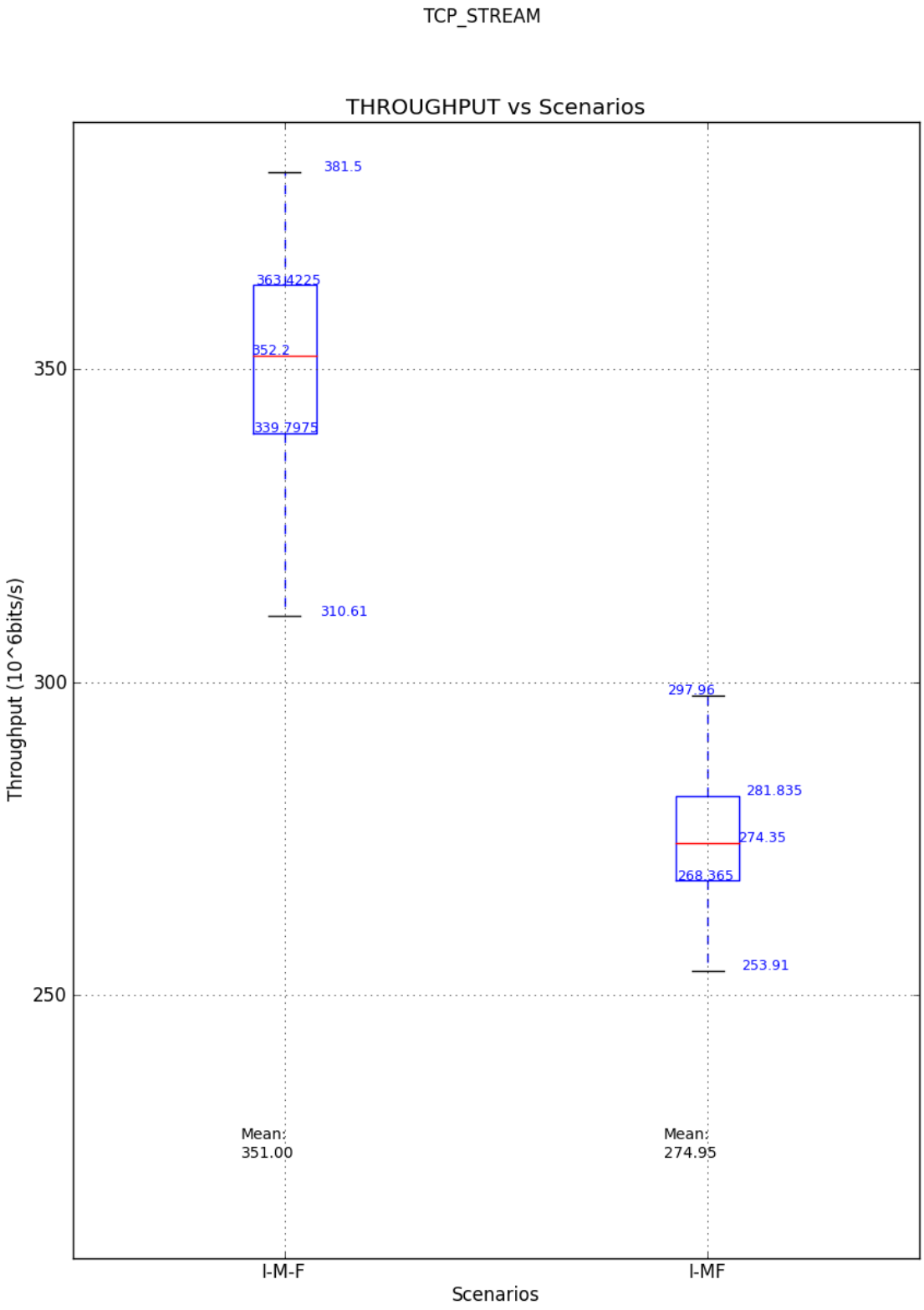

Figure E.16: Boxplots comparison of MYSEA processes using TCP_STREAM Tests

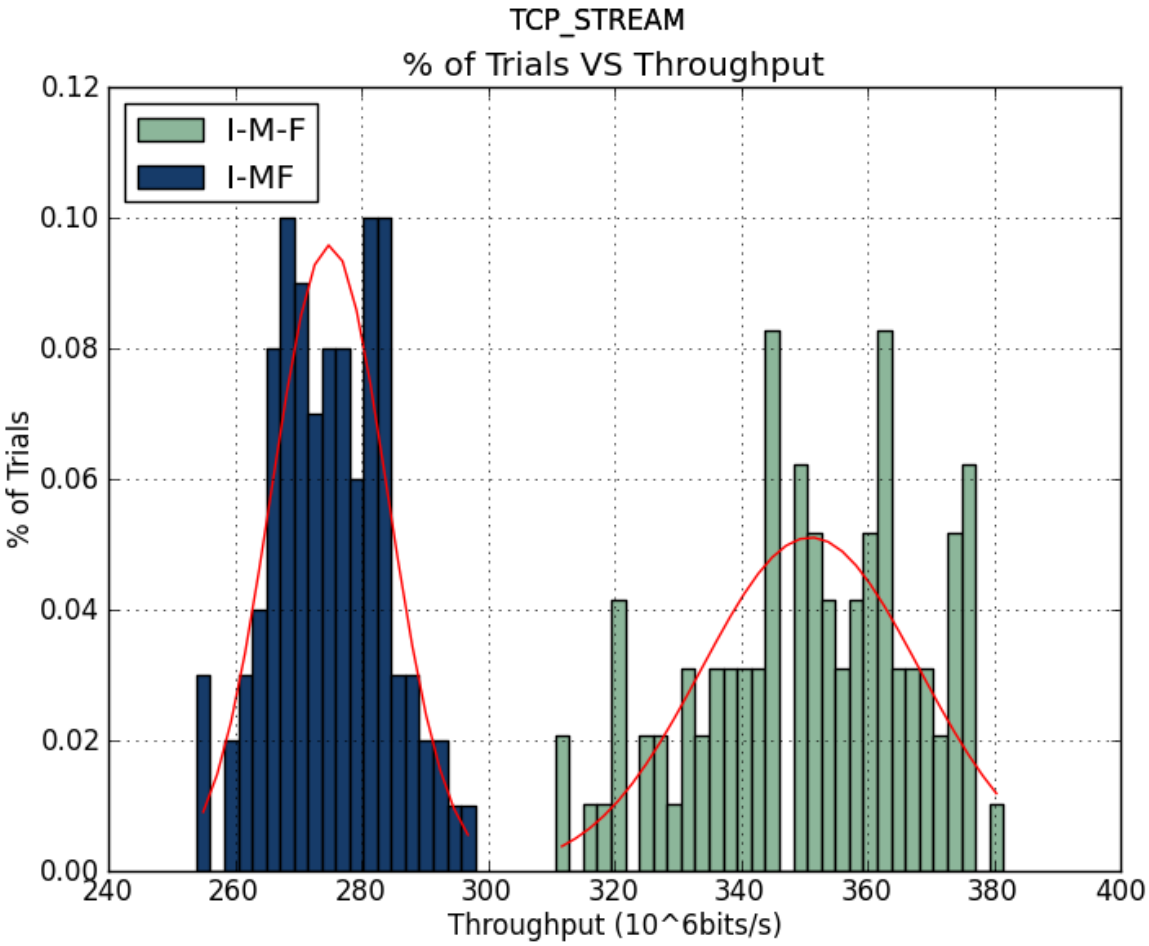

Figure E.17: Histogram comparison of MYSEA processes using TCP_STREAM Tests

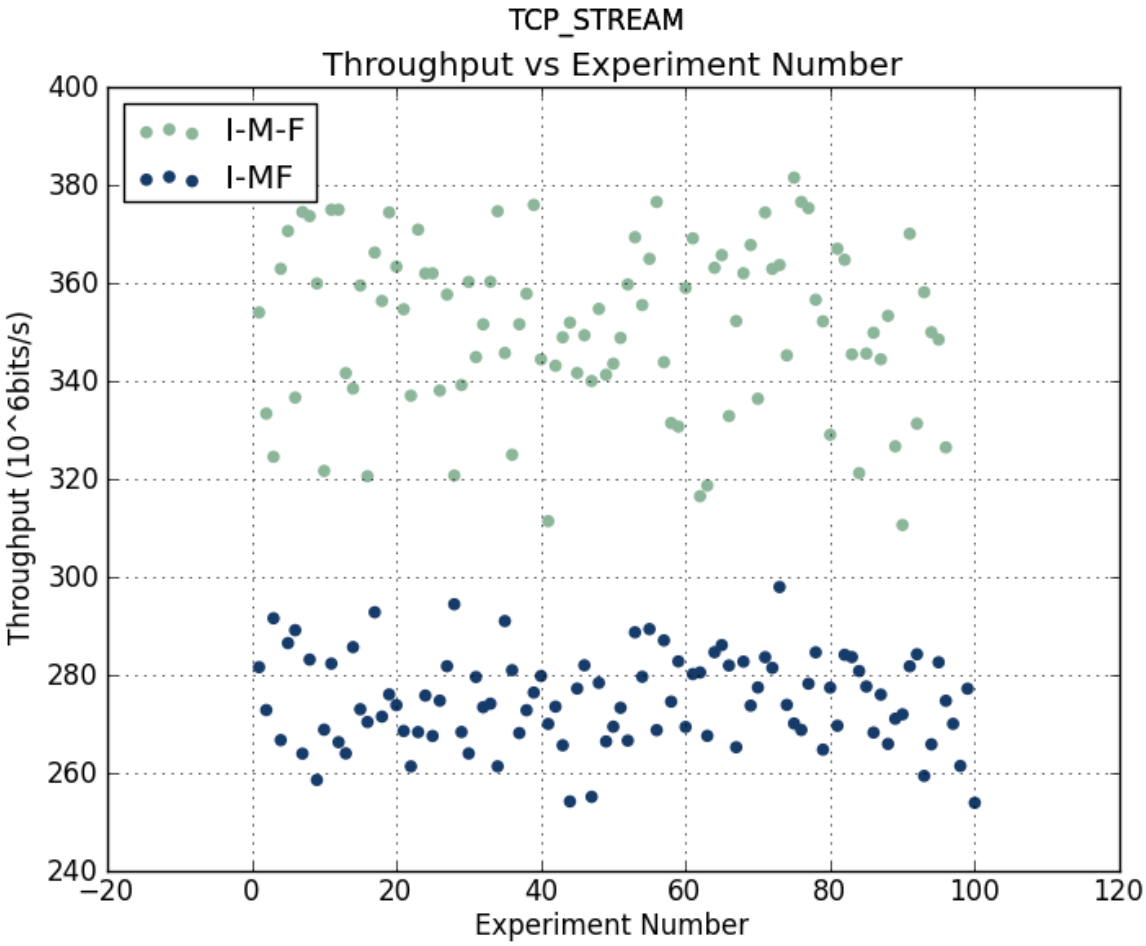

Figure E.18: Scatterplot comparison of MYSEA processes using TCP_STREAM Tests

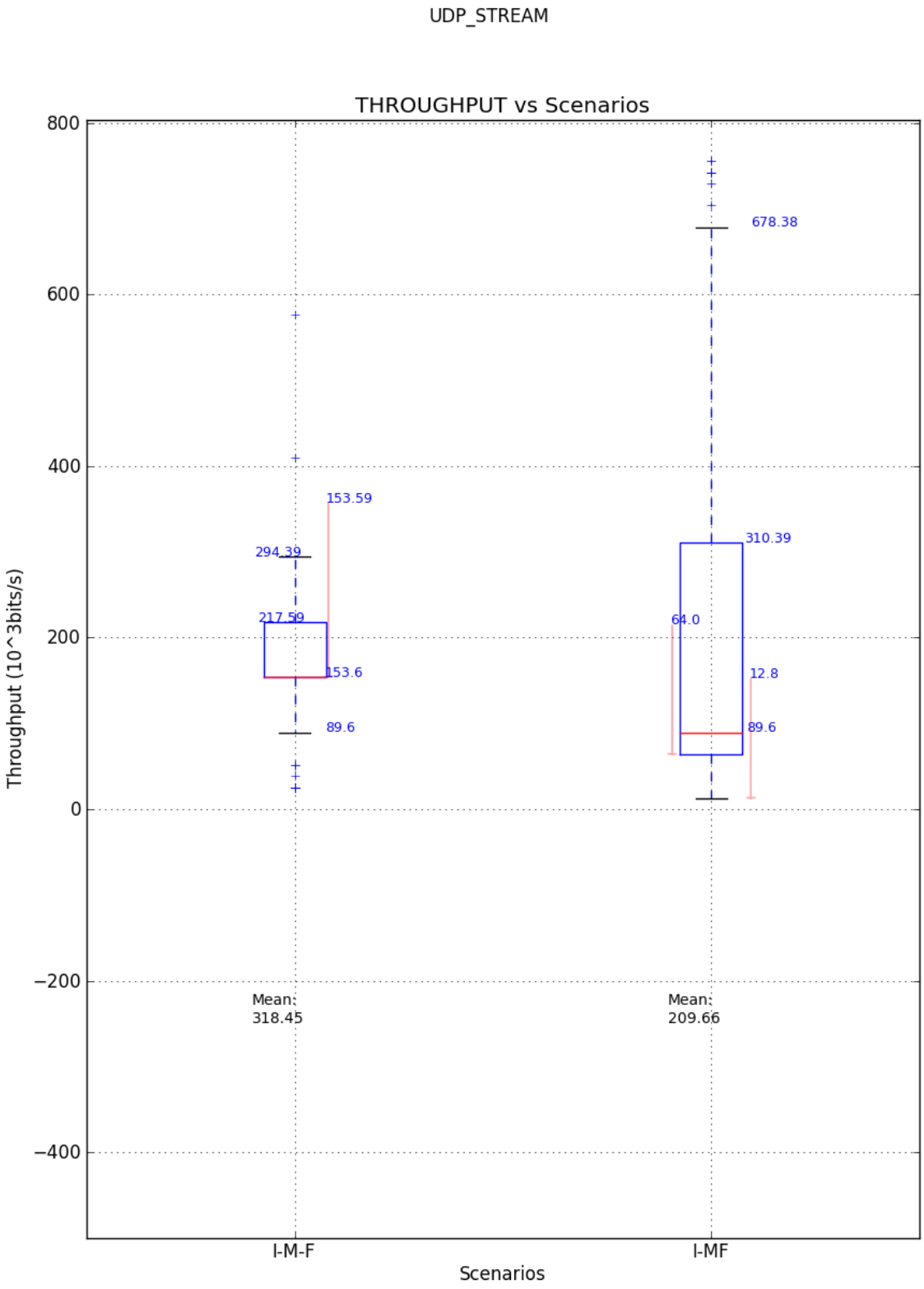

Figure E.19: Boxplots comparison of the performance of IPSec using UDP_STREAM Tests

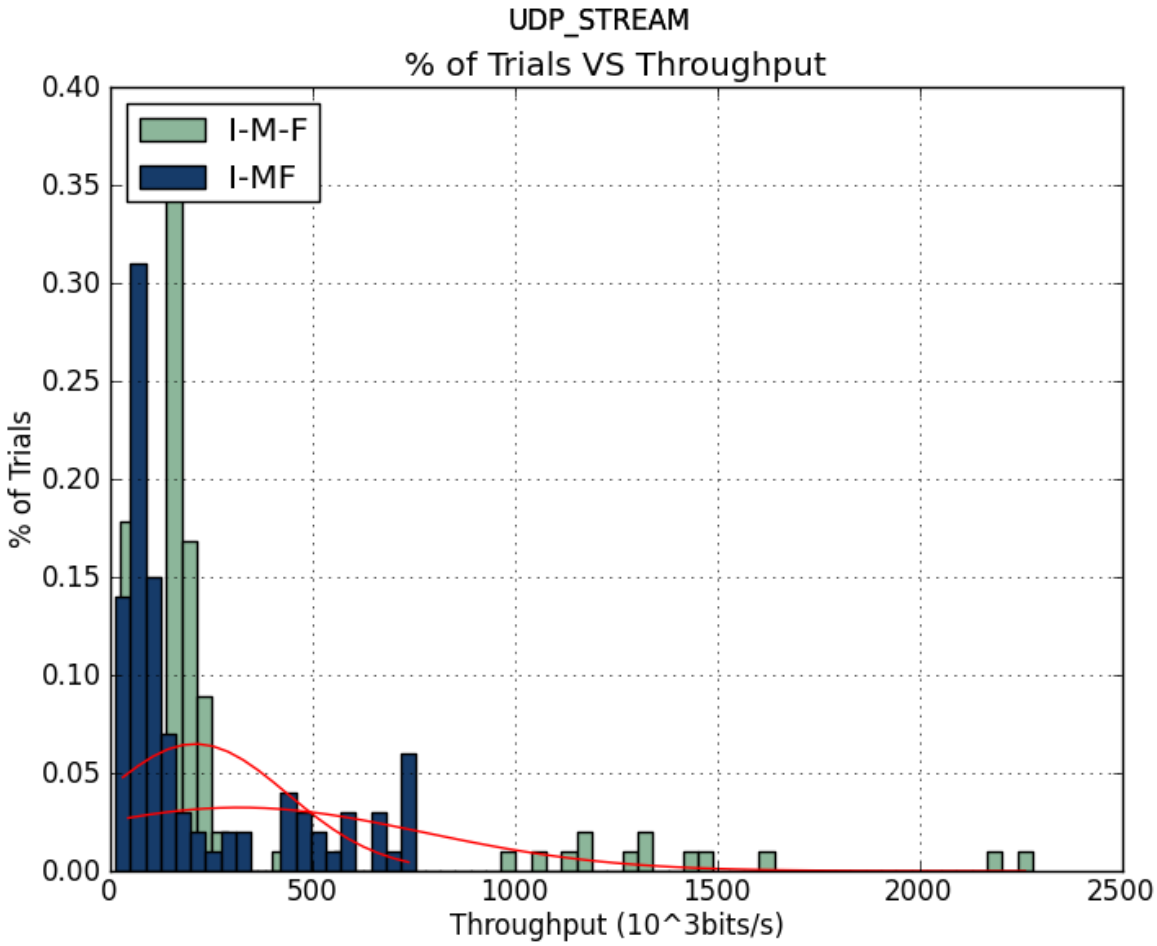

Figure E.20: Histogram comparison of the performance of IPSec using UDP_STREAM Tests

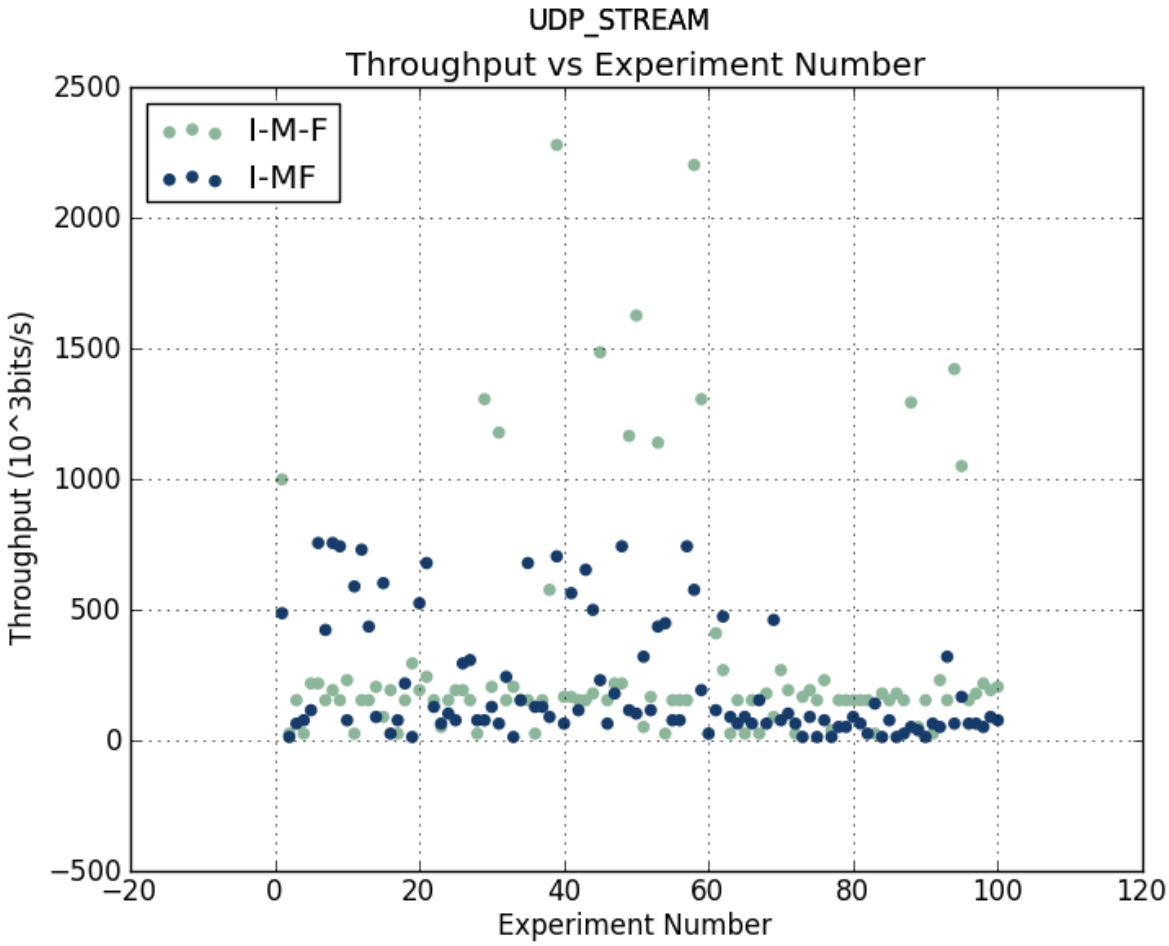

Figure E.21: Scatterplot comparison of the performance of IPSec using UDP_STREAM Tests

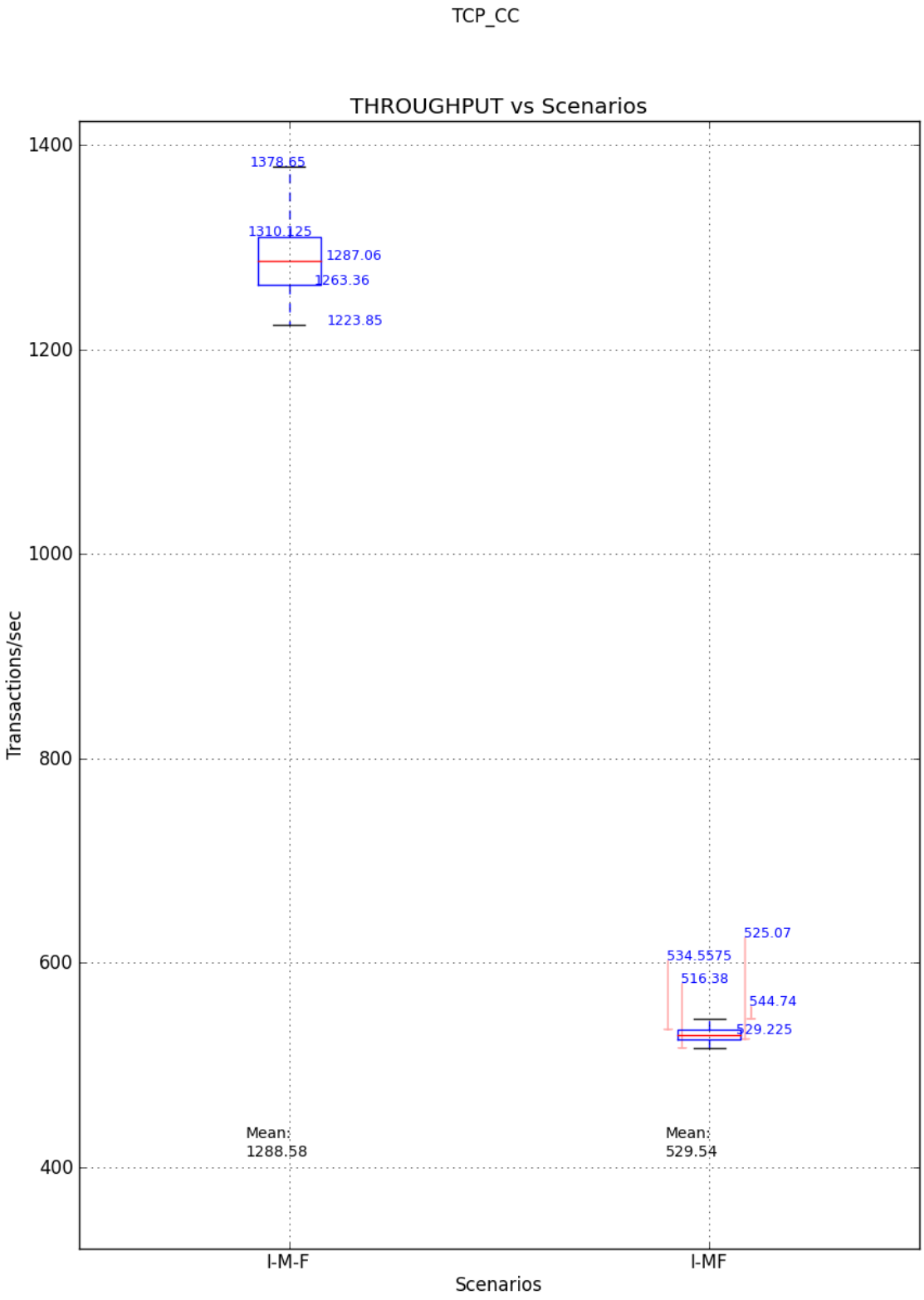

Figure E.22: Boxplots comparison of the performance of IPSec using TCP_CC Tests

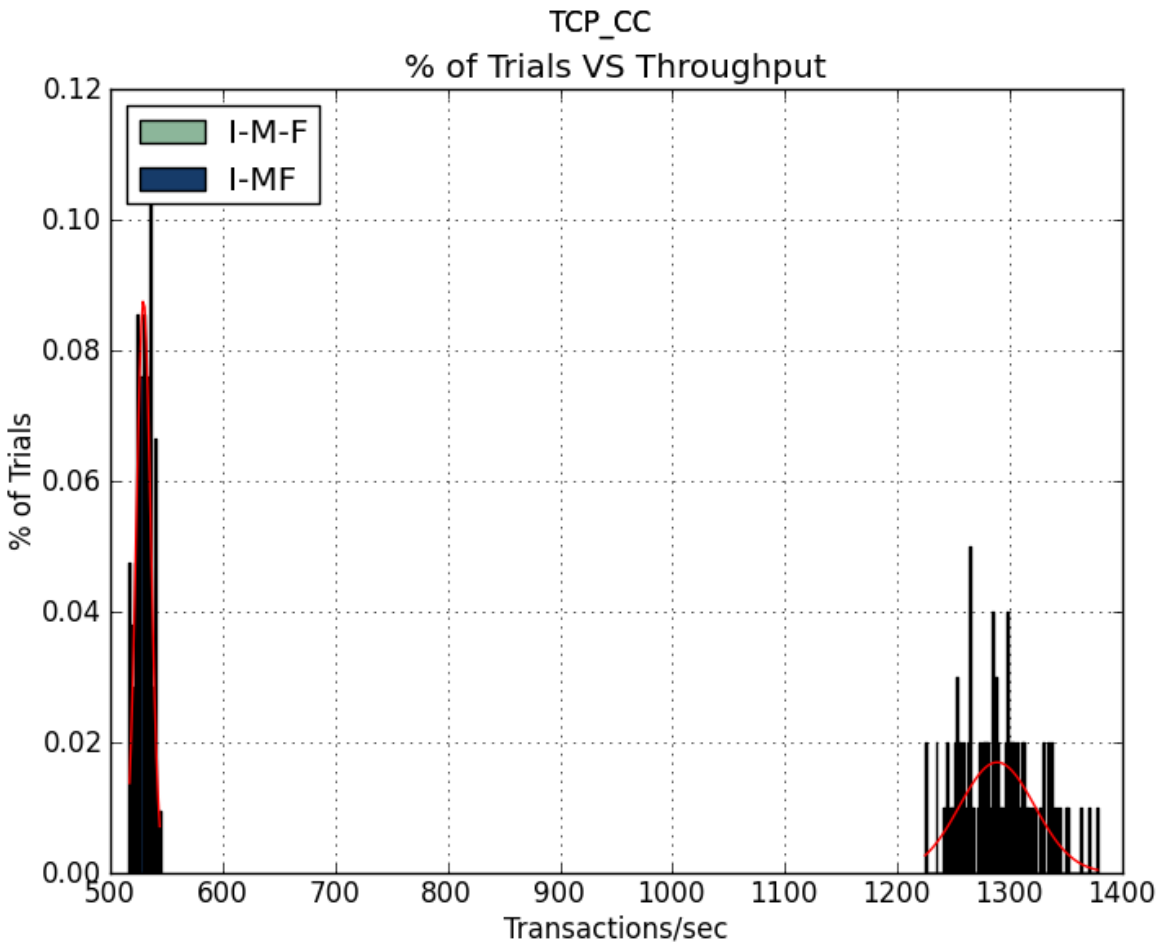

Figure E.23: Histogram comparison of the performance of IPSec using TCP_CC Tests

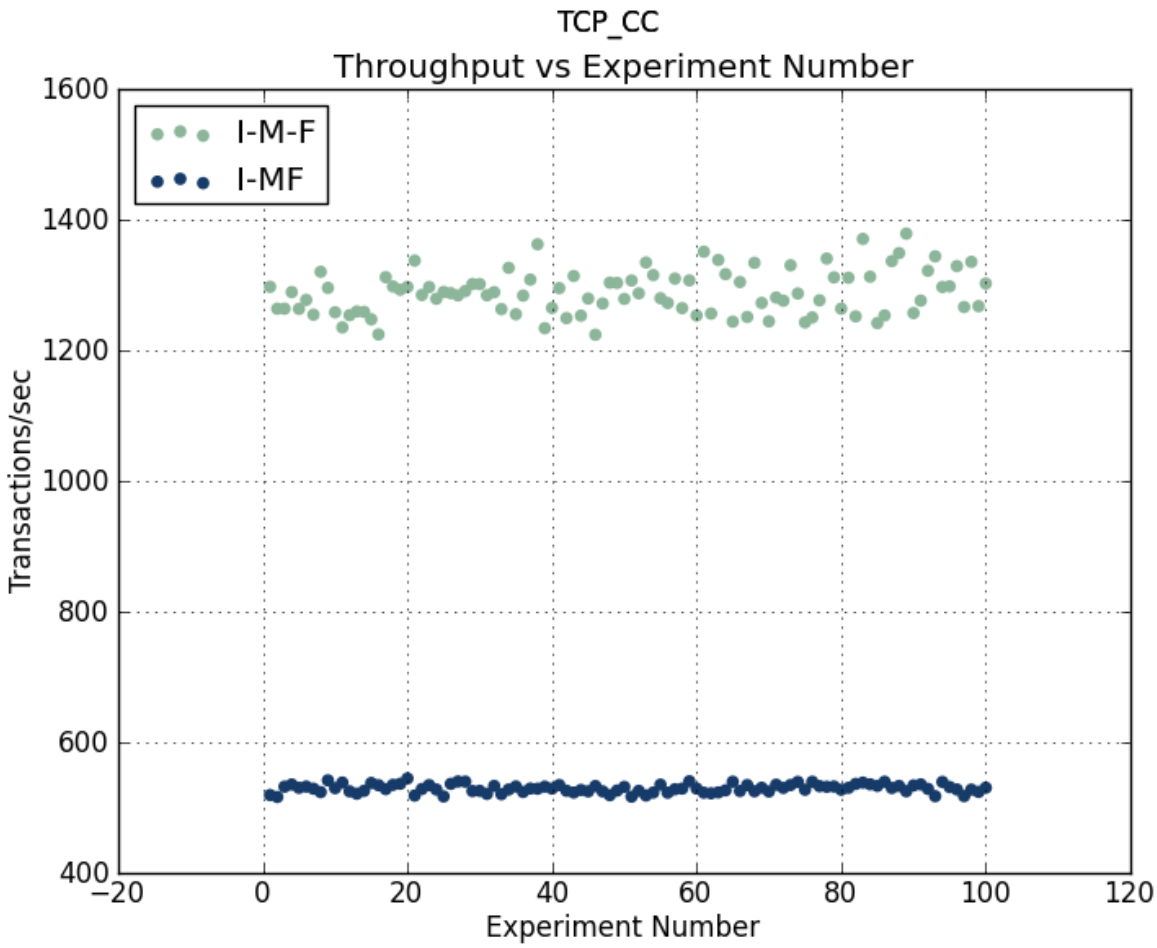

Figure E.24: Scatterplot comparison of the performance of IPSec using TCP_CC Tests

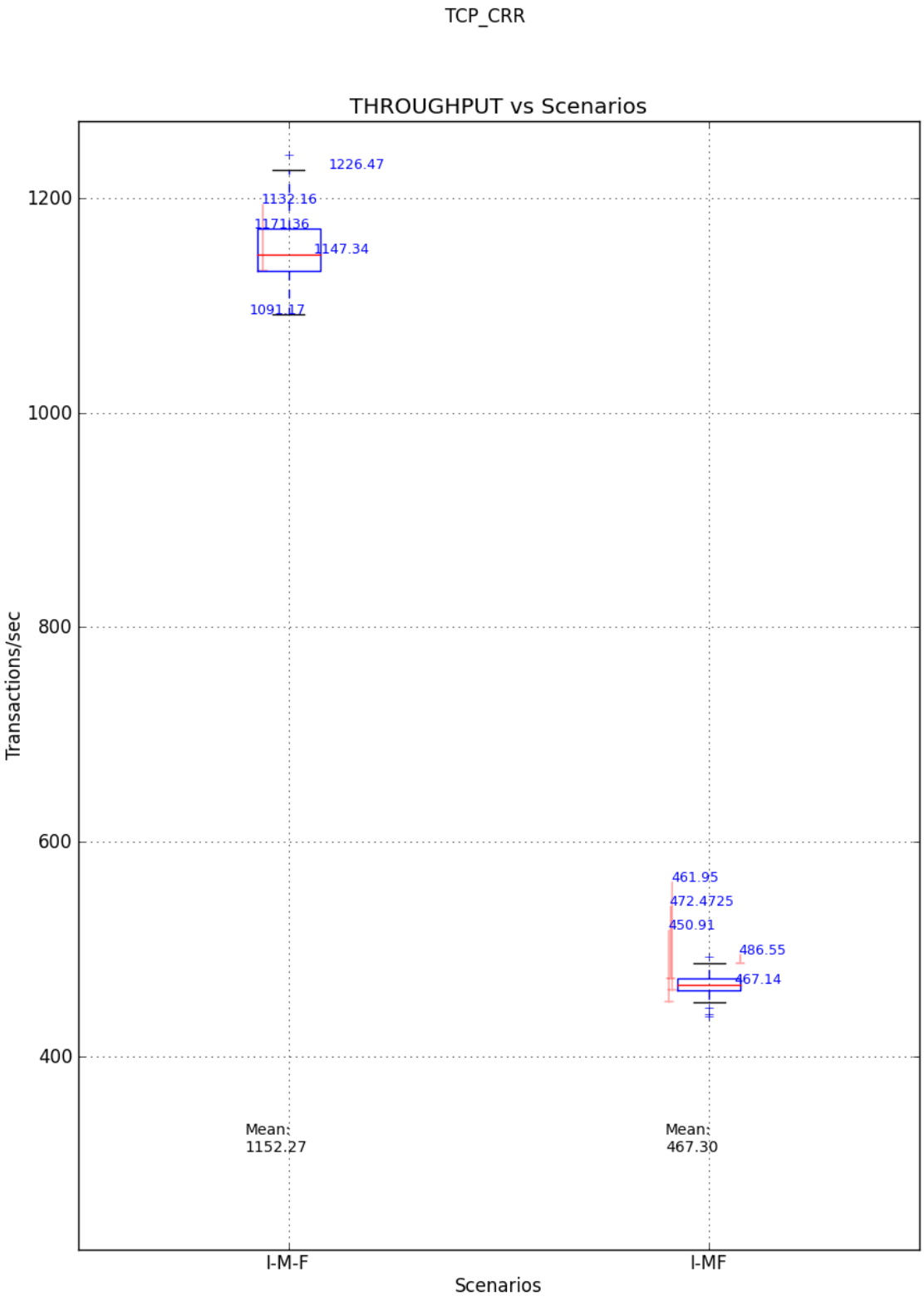

Figure E.25: Boxplots comparison of the performance of IPSec using TCP_CRR Tests

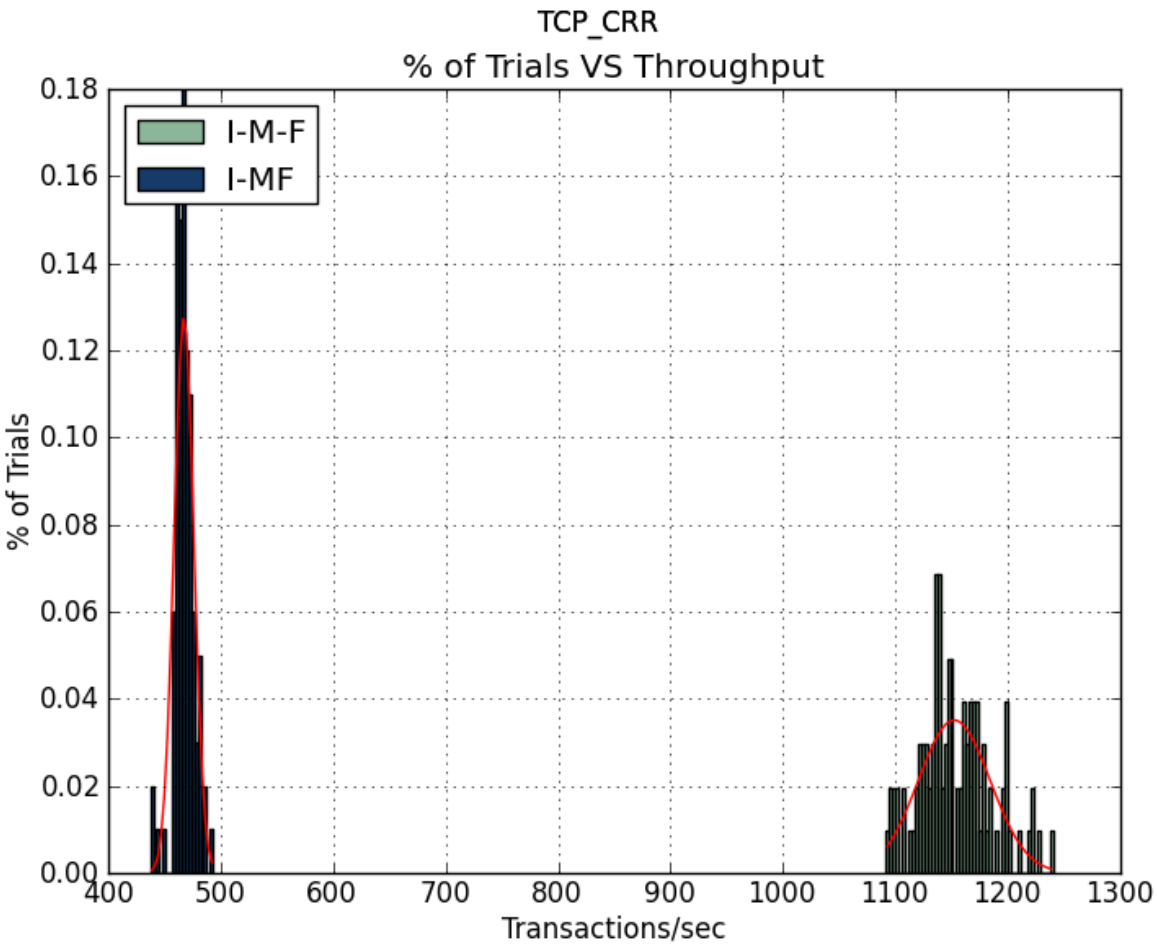

Figure E.26: Histogram comparison of the performance of IPSec using TCP_CRR Tests

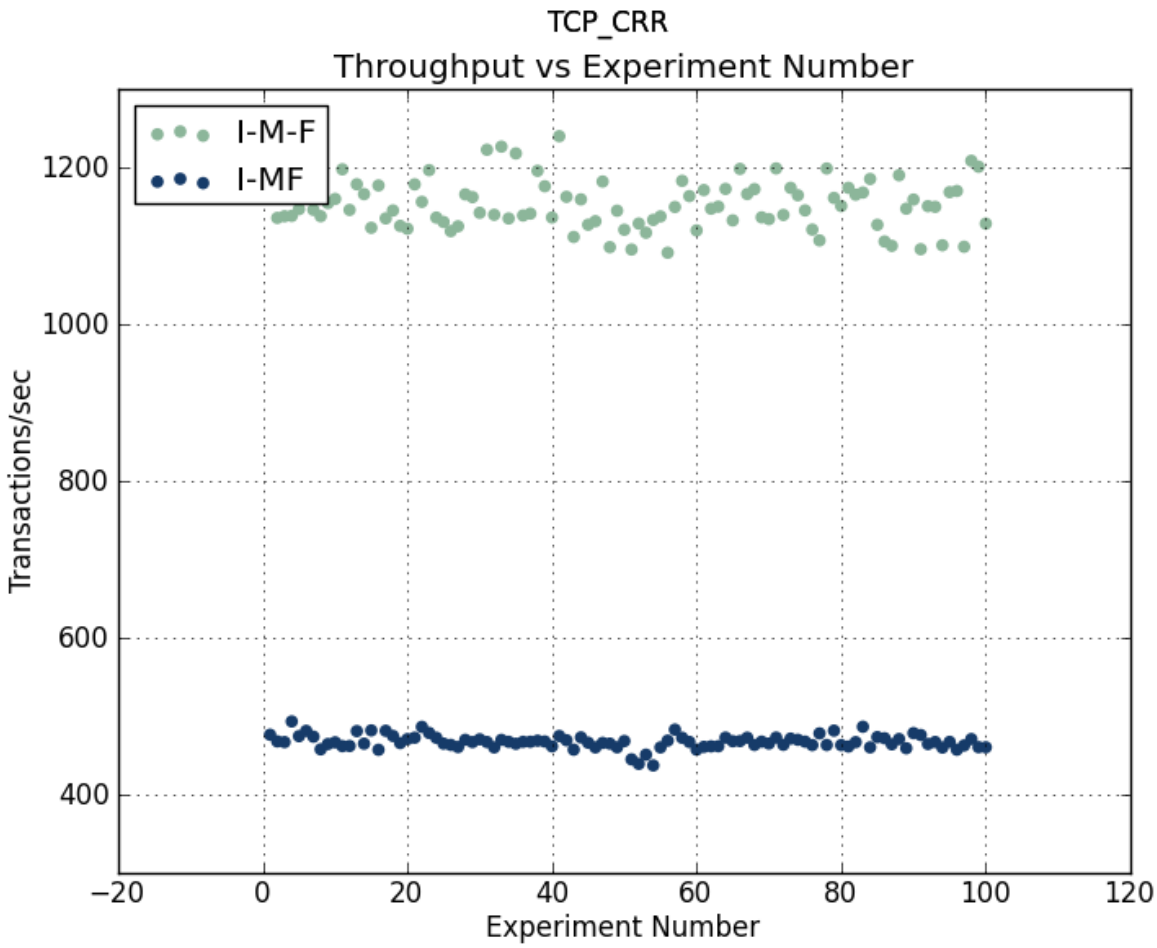

Figure E.27: Scatterplot comparison of the performance of IPSec using TCP_CRR Tests

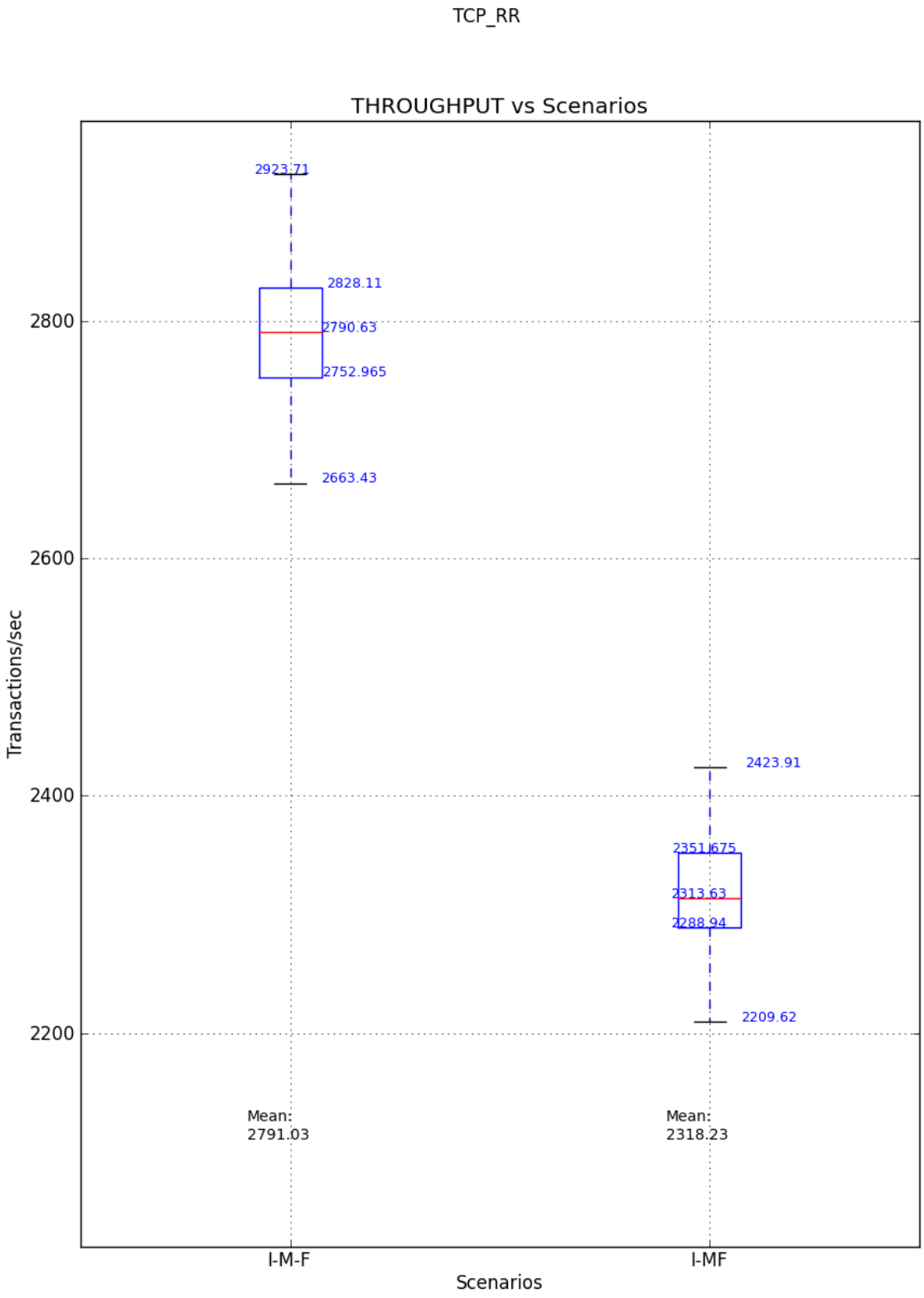

Figure E.28: Boxplots comparison of the performance of IPSec using TCP_RR Tests

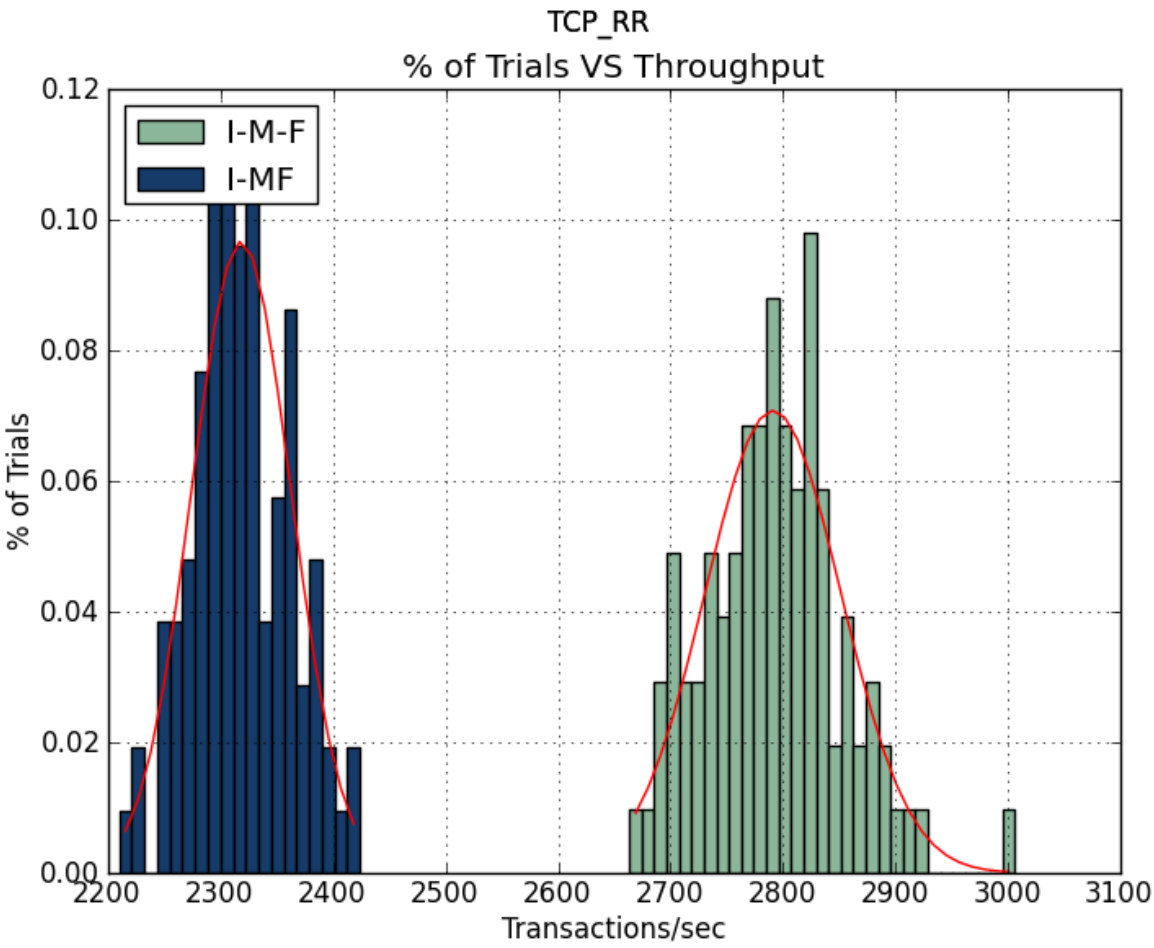

Figure E.29: Histogram comparison of the performance of IPSec using TCP_RR Tests

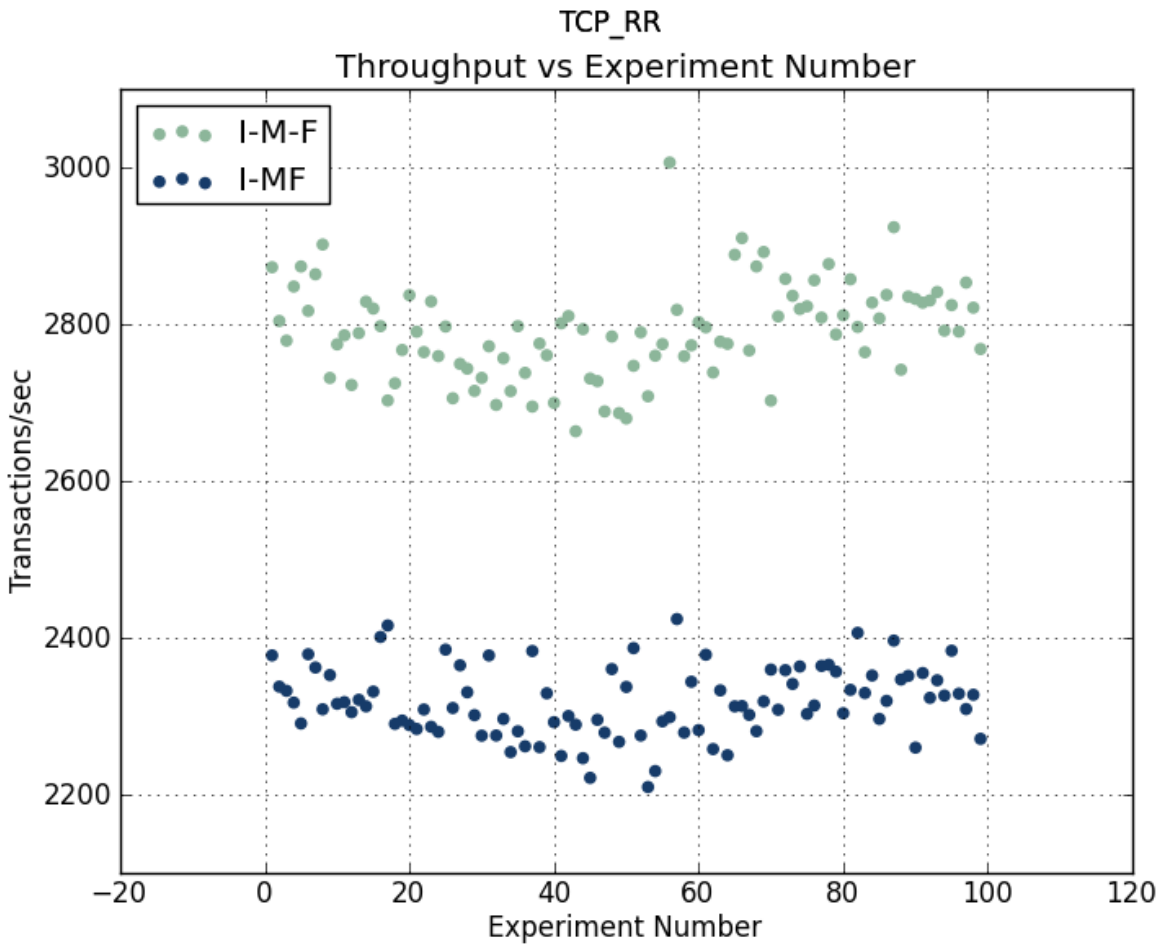

Figure E.30: Scatterplot comparison of the performance of IPSec using TCP_RR Tests

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F:
Results of APS and TPS on different servers

This appendix contains the results of the benchmark tests used for the analysis of the performance overhead of the MYSEA Federation.

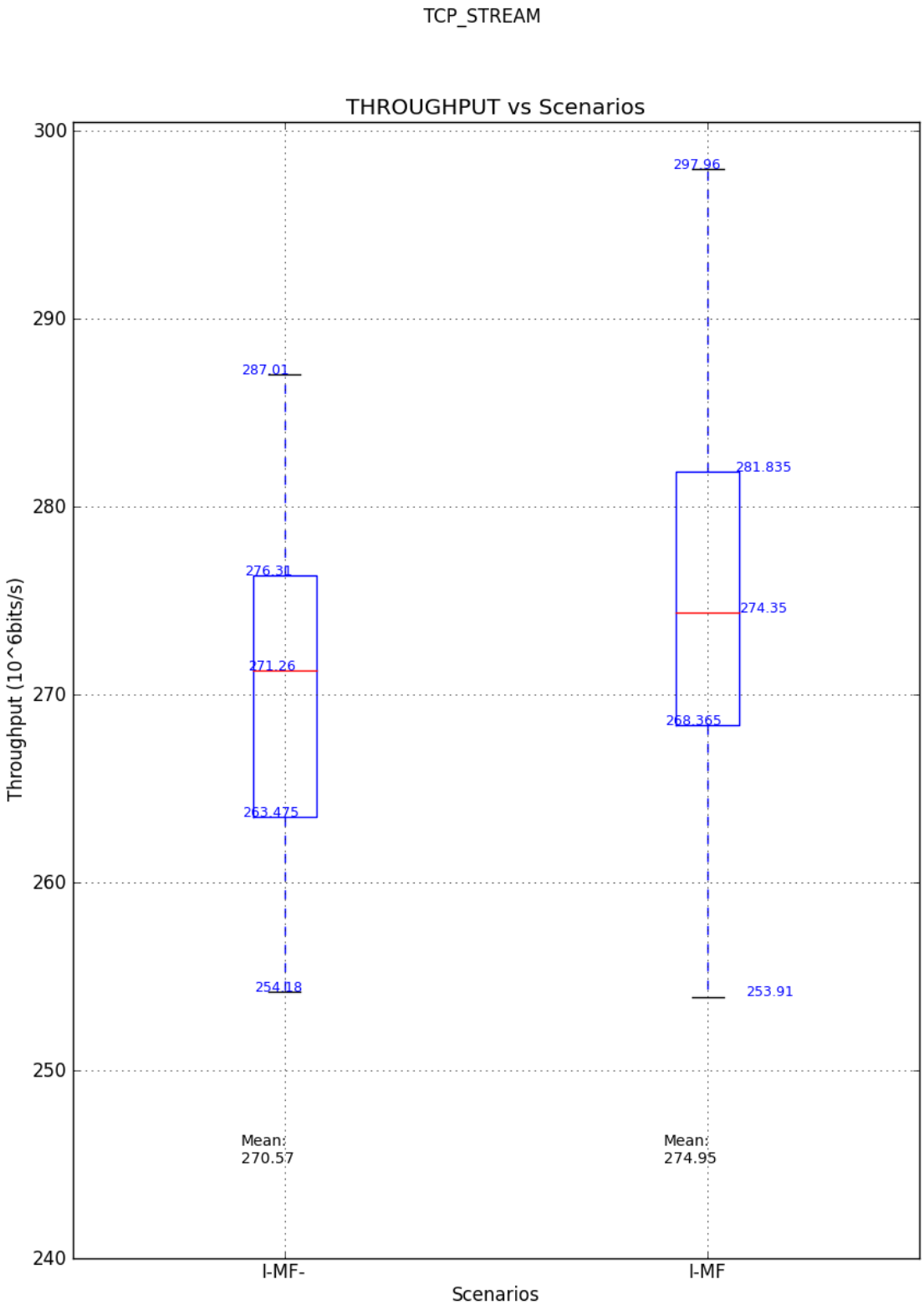

Figure F.1: Boxplots comparison of APS and TPS on different servers using TCP_STREAM Tests

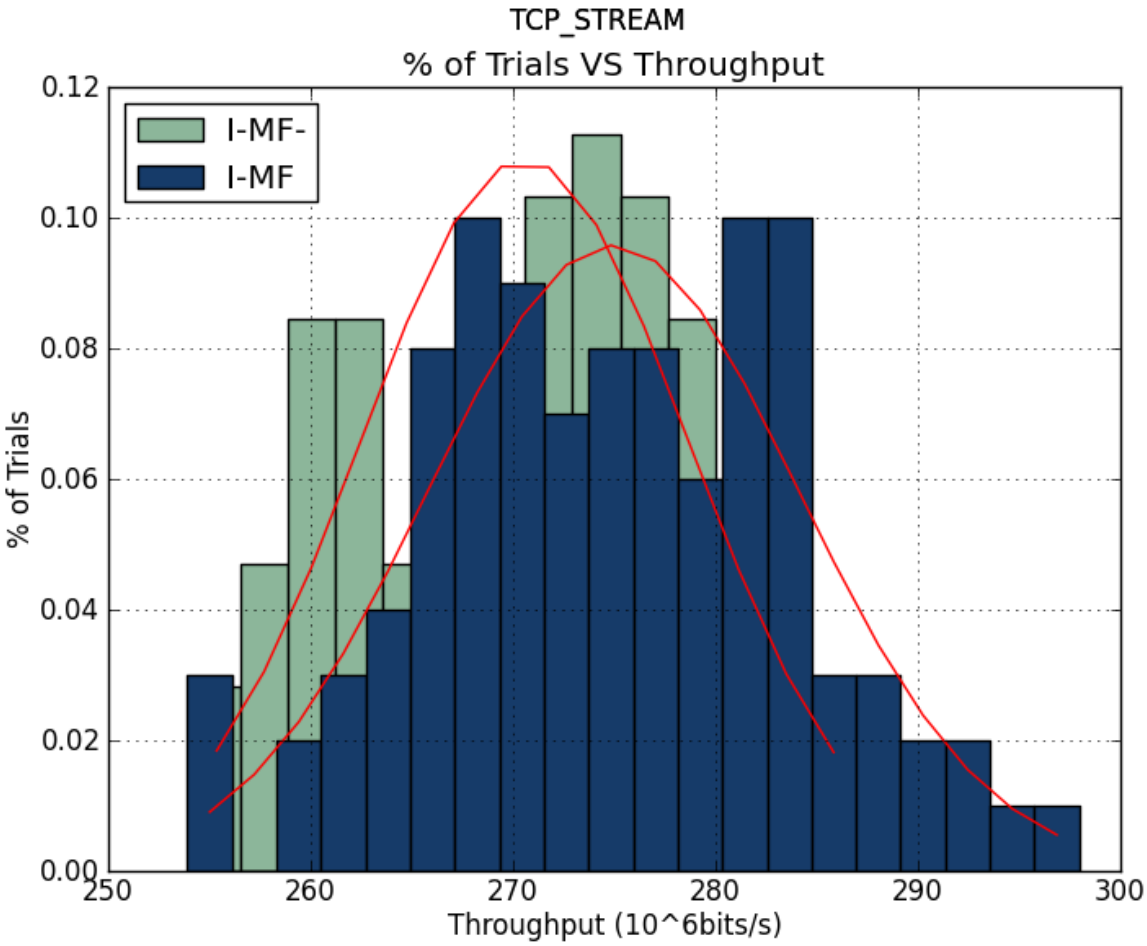

Figure F.2: Histograms of APS and TPS on different servers using TCP_STREAM Tests

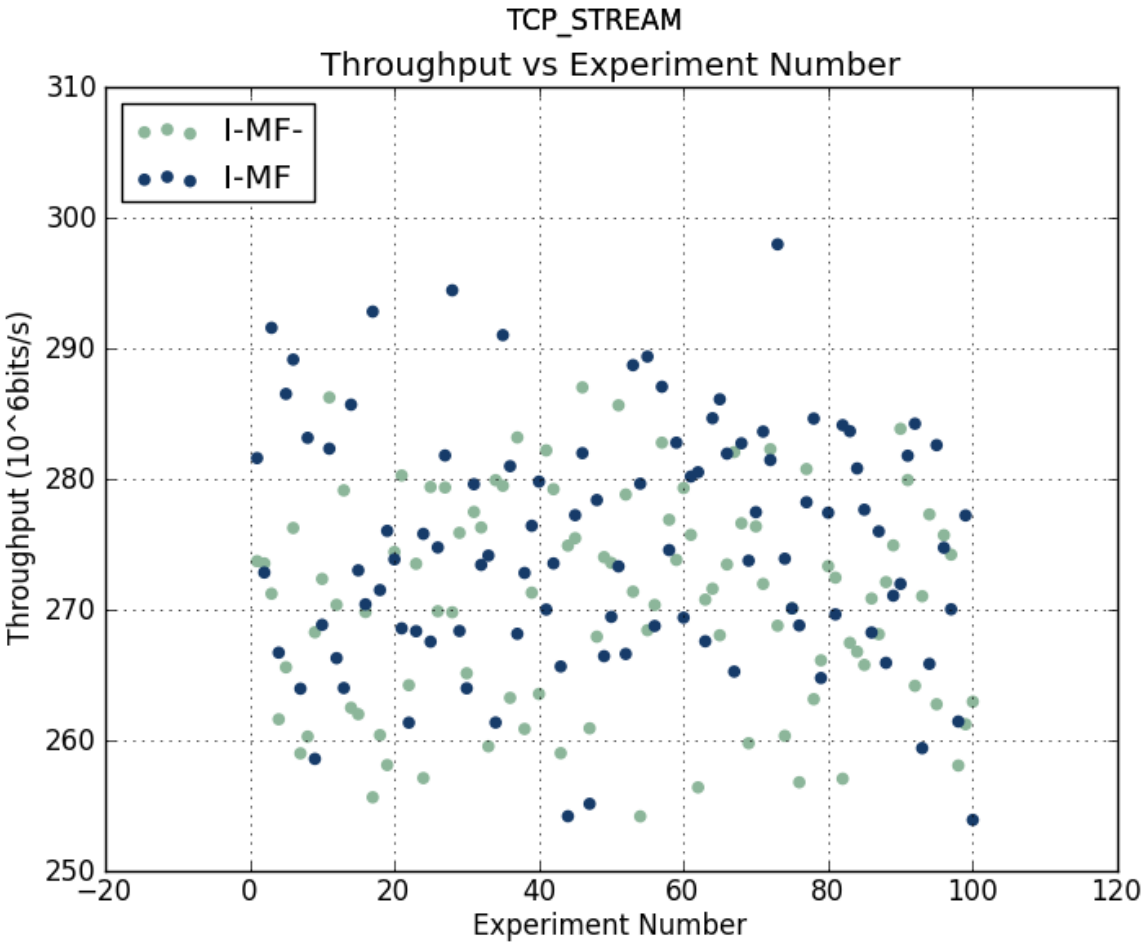

Figure F.3: Scatterplots of APS and TPS on different servers using TCP_STREAM Tests

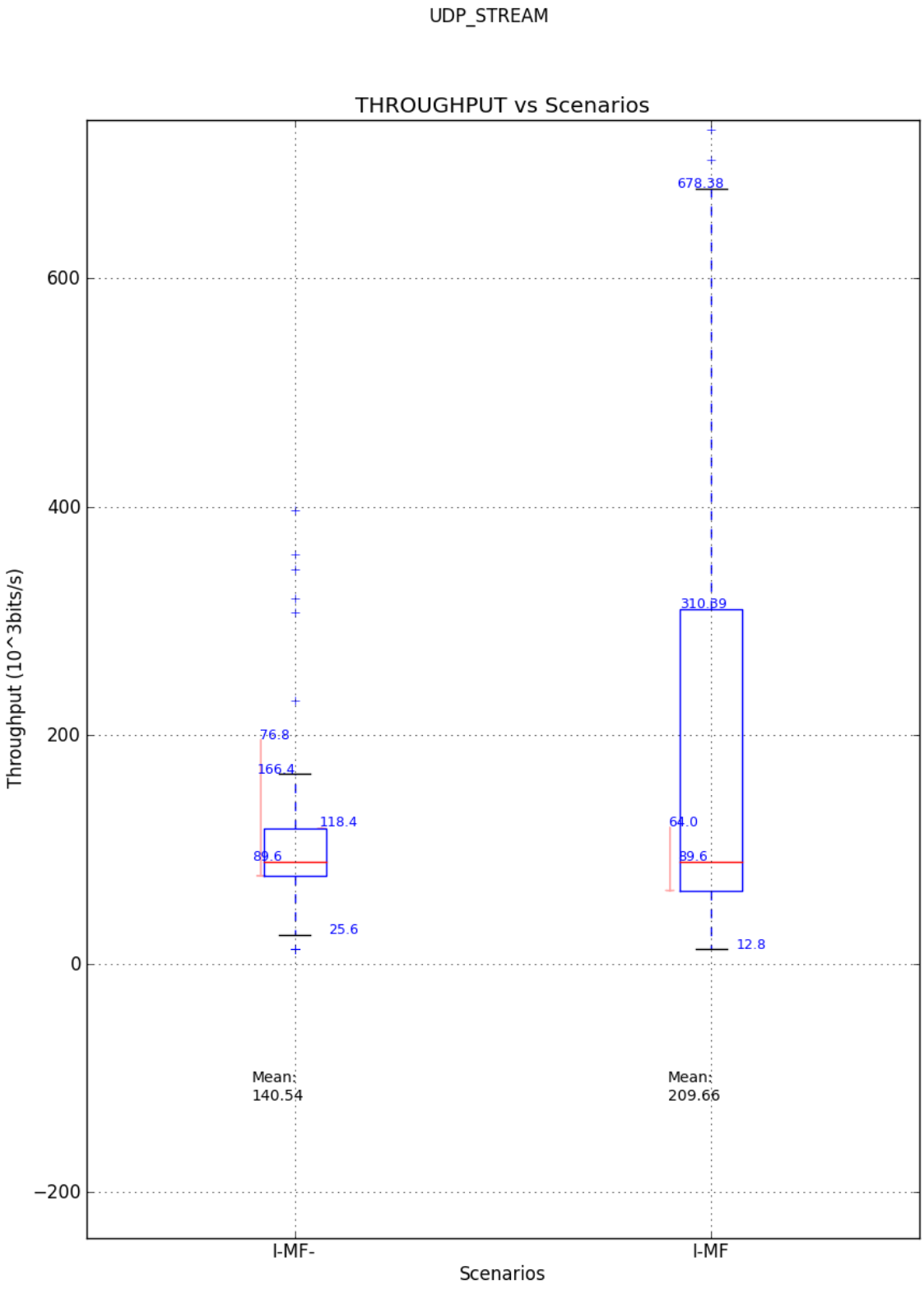

Figure F.4: Boxplots comparison of APS and TPS on different servers using UDP_STREAM Tests

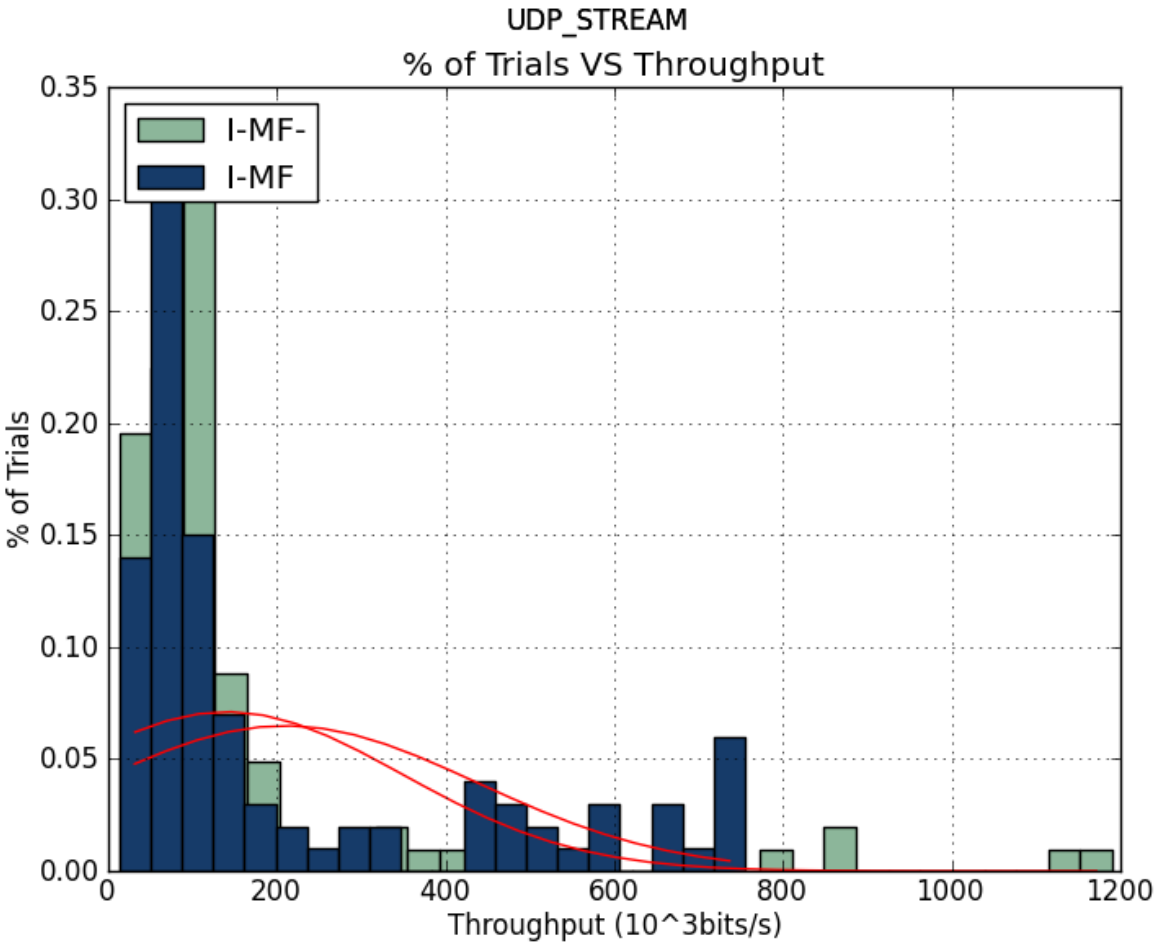

Figure F.5: Histograms of APS and TPS on different servers using UDP_STREAM Tests

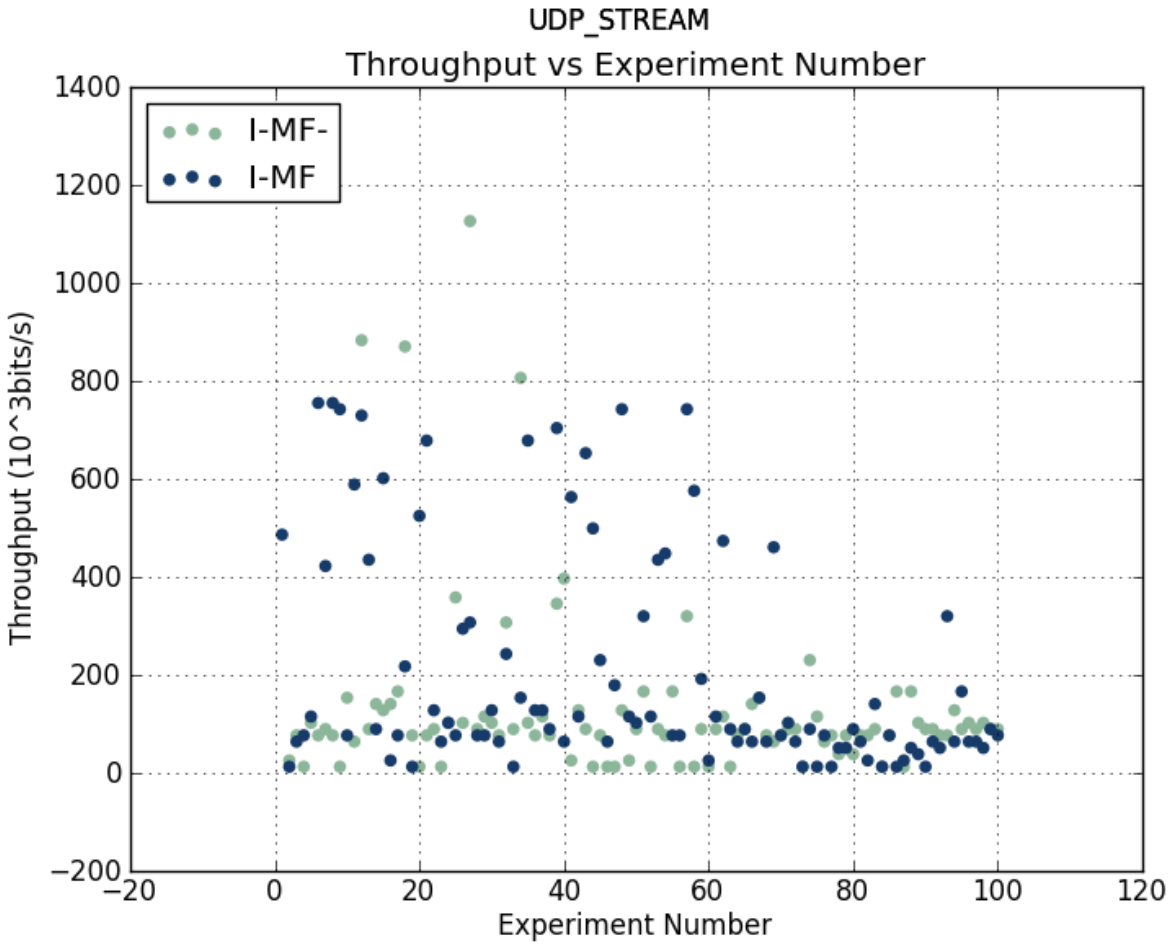

Figure F.6: Scatterplots comparison of APS and TPS on different servers using UDP_STREAM Tests

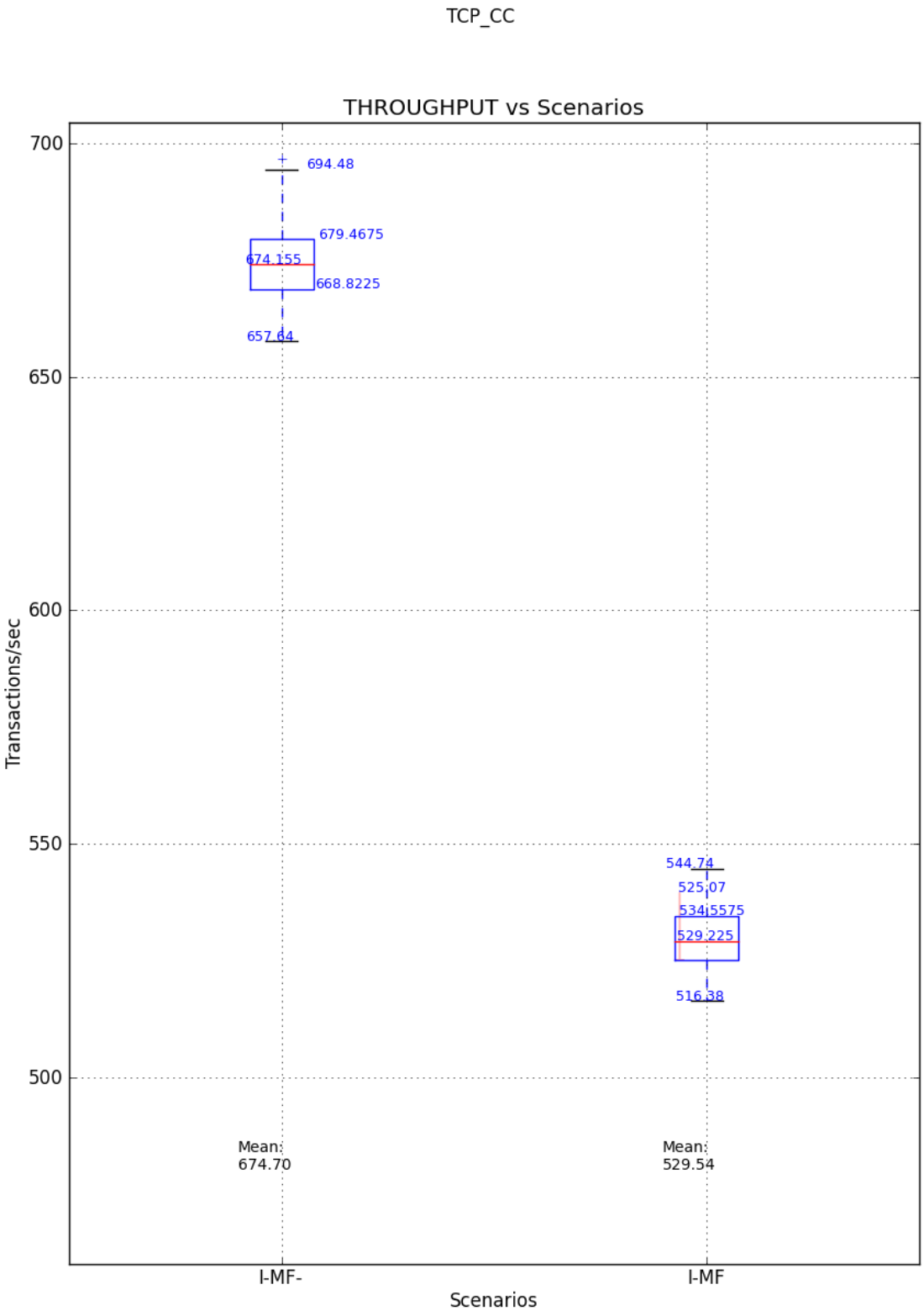

Figure F.7: Boxplots comparison of APS and TPS on different servers using TCP_CC Tests

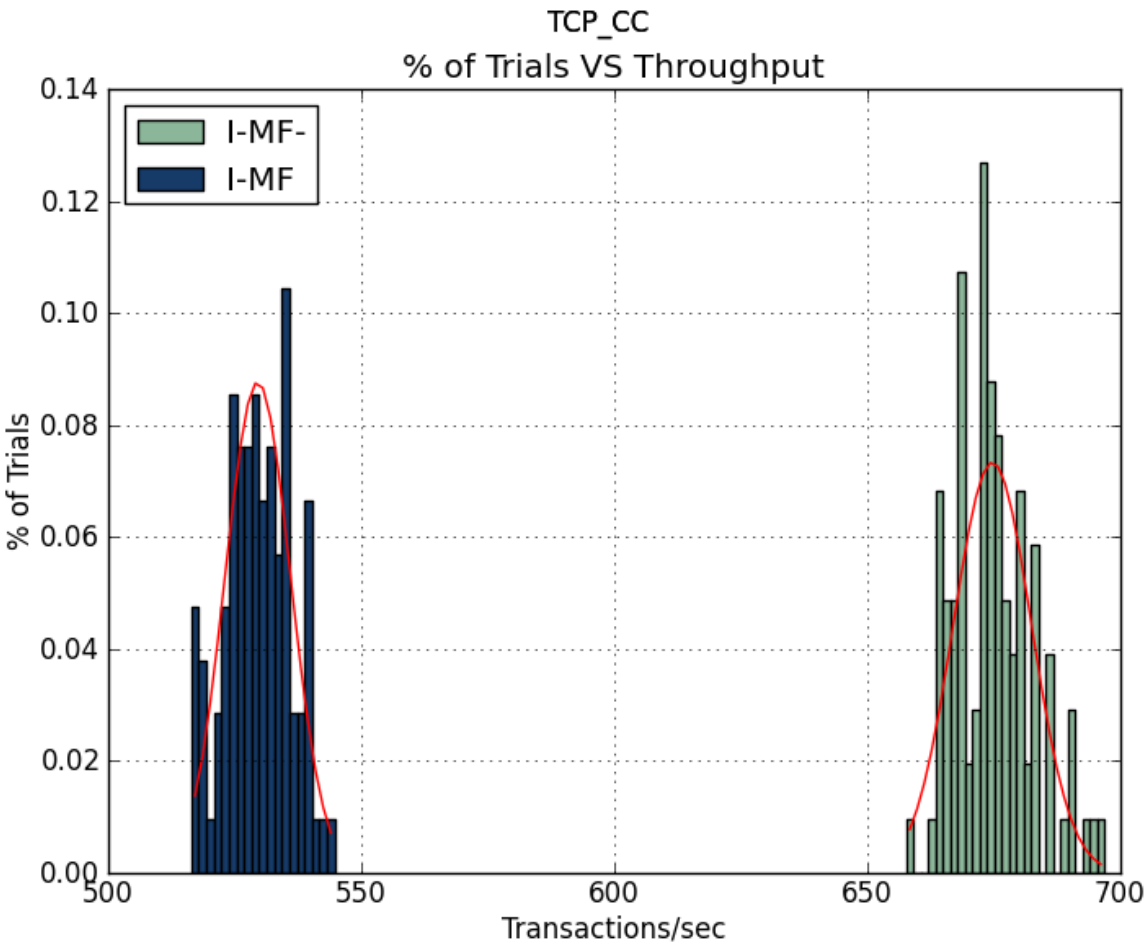

Figure F.8: Histograms of APS and TPS on different servers using TCP_CC Tests

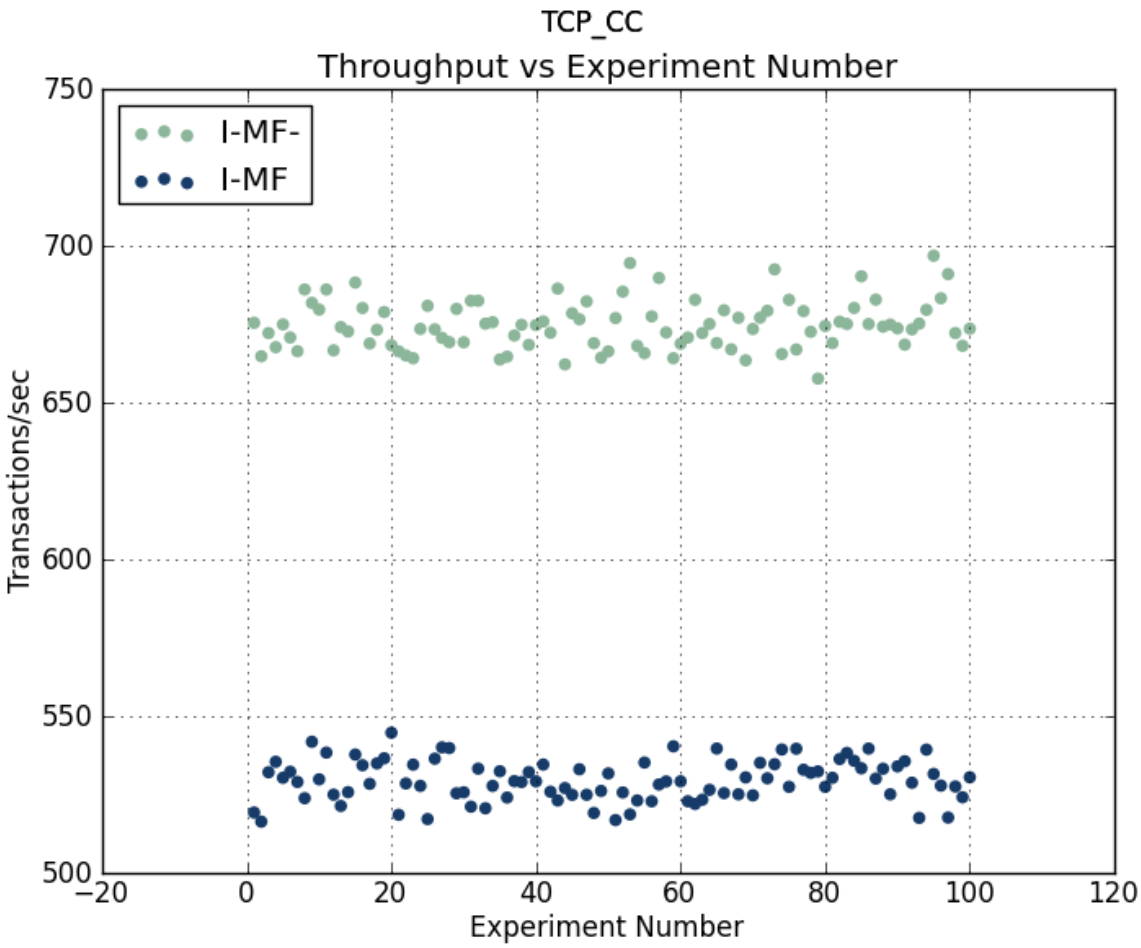

Figure F.9: Scatterplots comparison of APS and TPS on different servers using TCP_CC Tests

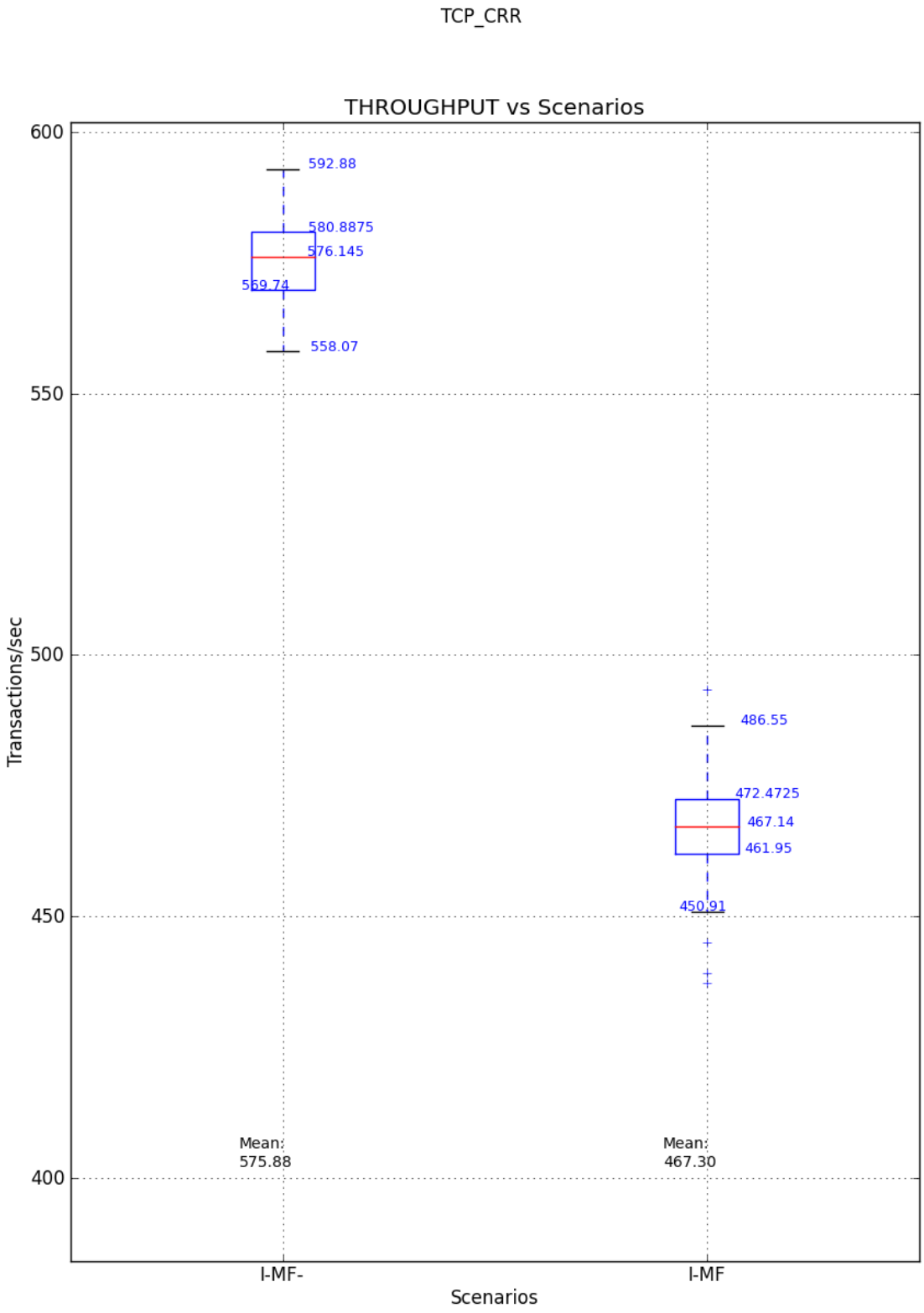

Figure F.10: Boxplots comparison of APS and TPS on different servers using TCP_CRR Tests

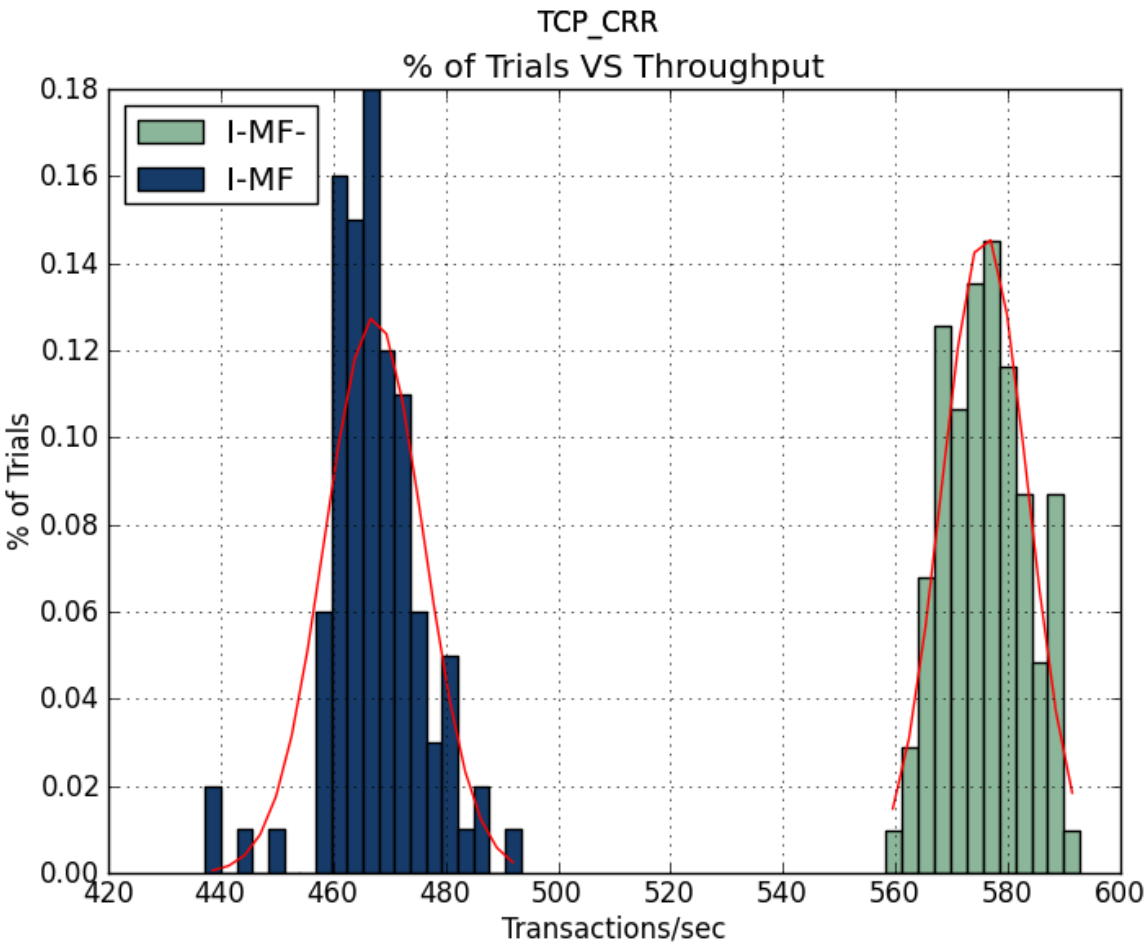

Figure F.11: Histograms of APS and TPS on different servers using TCP_CRR Tests

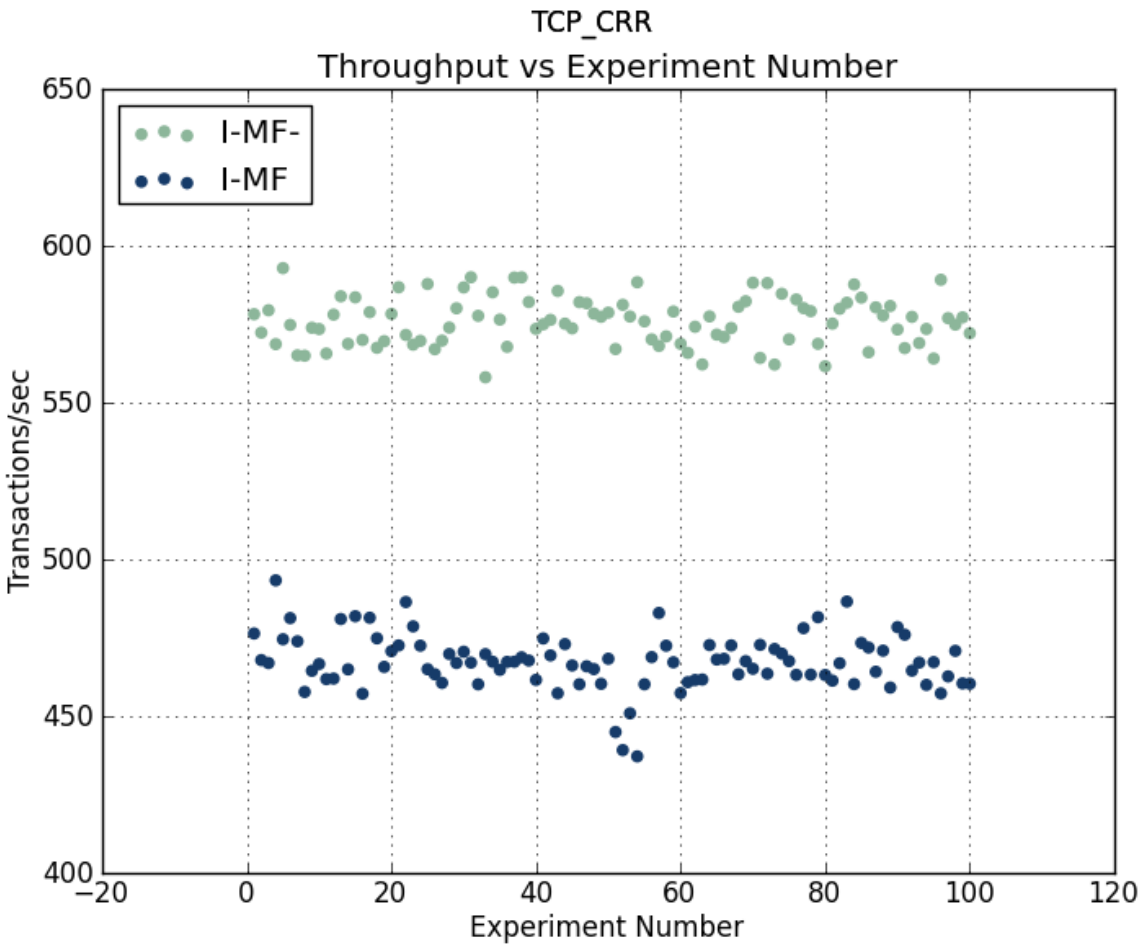

Figure F.12: Scatterplots comparison of APS and TPS on different servers using TCP_CRR Tests

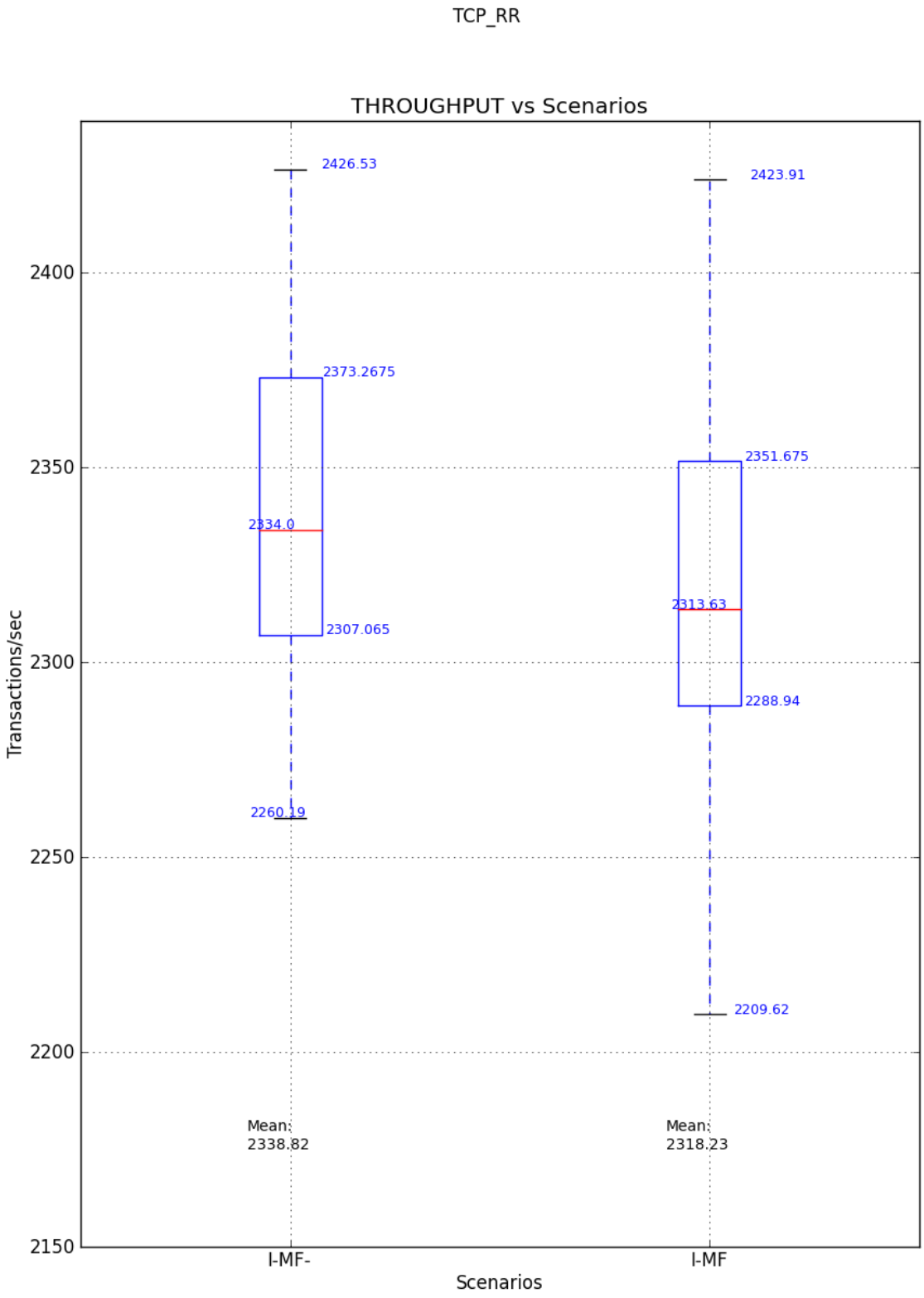

Figure F.13: Boxplots comparison of APS and TPS on different servers using TCP_RR Tests

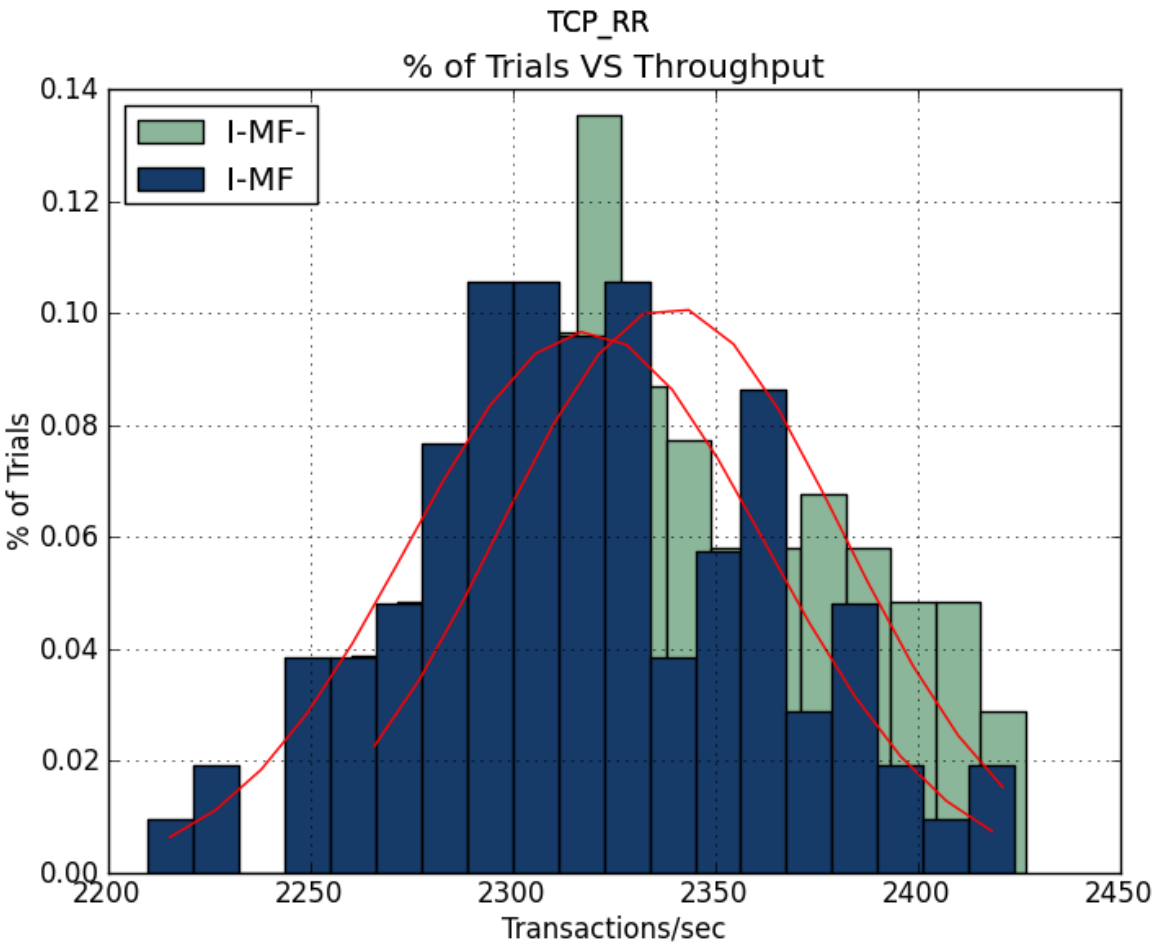

Figure F.14: Histograms of APS and TPS on different servers using TCP_RR Tests

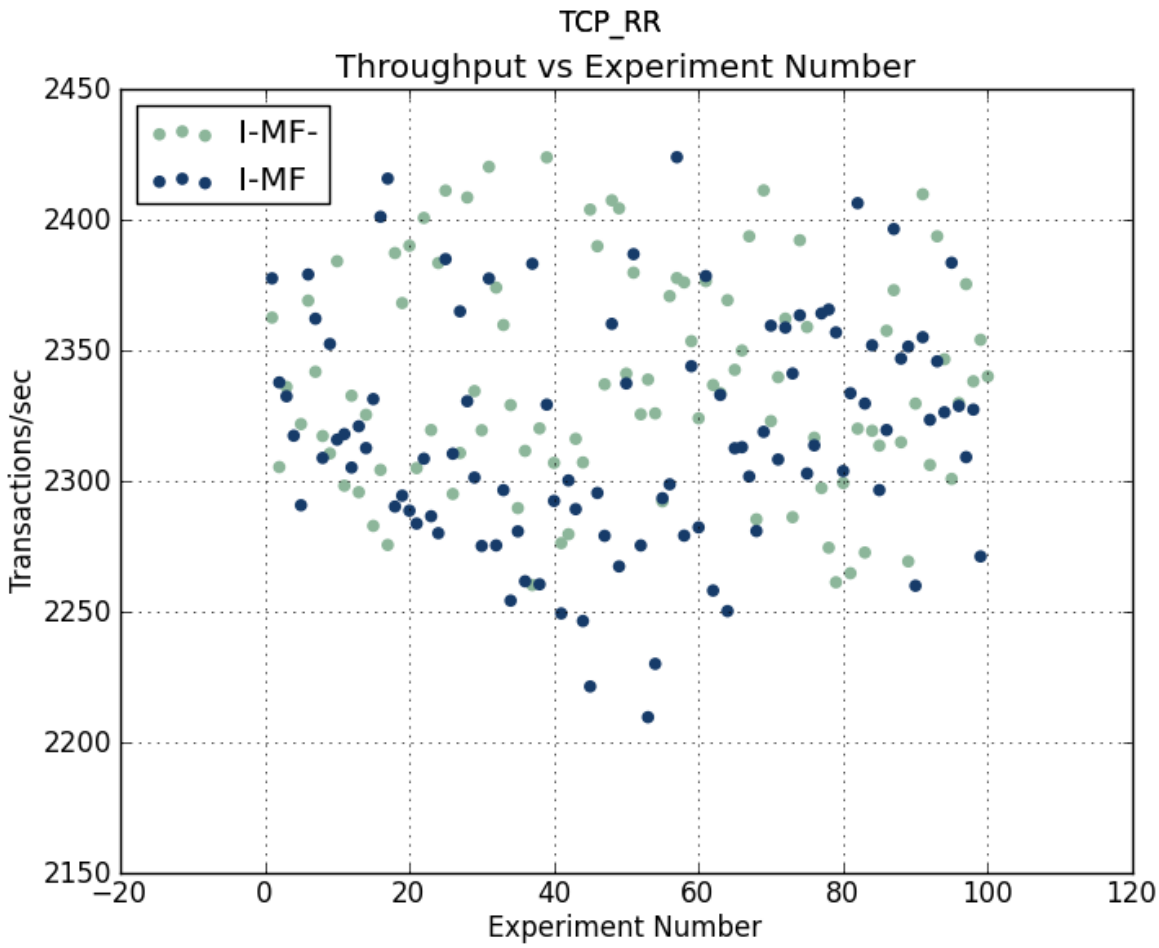

Figure F.15: Scatterplots comparison of APS and TPS on different servers using TCP_RR Tests

APPENDIX G:
UDP Discussion

The results of UDP_STREAM test show abnormalities in its throughput. Figure G.1 shows the boxplot summary of test results for UDP_STREAM test. The throughput mean values for the scenarios when IPSec encryption tunnel is not deployed is smaller than when the IPSec encryption tunnel is deployed. This is counter intuitive as the presence of IPSec encryption tunnel encrypting UDP packets should result in a additional overhead, and thus reduce its throughput. Upon analysis of the raw test data for the scenario when NO IPSec encryption is used, we discovered that the throughput is low because of the large disparity between the number of udp packets sent by netperf and the number of udp packets received by netserver. Table G.1 shows the number of udp packets sent and received when no IPSec encryption is used.

In addition, Table G.2 shows the test results for the scenario when IPSec encryption is used. We see a greater than 50% decrease in the number of UDP packets received.

Table G.1: Examples of UDP_STREAM Test raw data for NO IPSec encryption

No.	local send throughput ($10^3 bits/s$)	remove recv throughput ($10^3 bits/s$)	local packets sent	remote packets recv
1	960418.23	998.37	75035	78
2	960815.57	25.6	75066	2
3	960764.19	153.6	75062	12
4	960760.61	25.6	75062	2
5	.....			
6	.....			

The following are the suggested steps for future work.

1. Use a network analyzer to capture the UDP packets Verify that the number of packets captured on both sending and receiving party is accurately reported by netserver.
2. Analyze the function in source code: recv_omni in nettest_omni.c An error message was reported in this function: " YO! TIMESUP!". Verify that the system calls that netserver use for UDP related tests are valid in STOP OS.

Table G.2: Examples of UDP_STREAM Test raw data when IPSec encryption is used

No.	local send throughput ($10^3 bits/s$)	remove recv throughput ($10^3 bits/s$)	local packets sent	remote packets recv
1	326865.51	141674.15	25538	11069
2	327094.31	155394.12	25556	12141
3	329015.93	149571.31	25706	11686
4	327903.53	149546.23	25619	11684
5	.....			
6	.....			

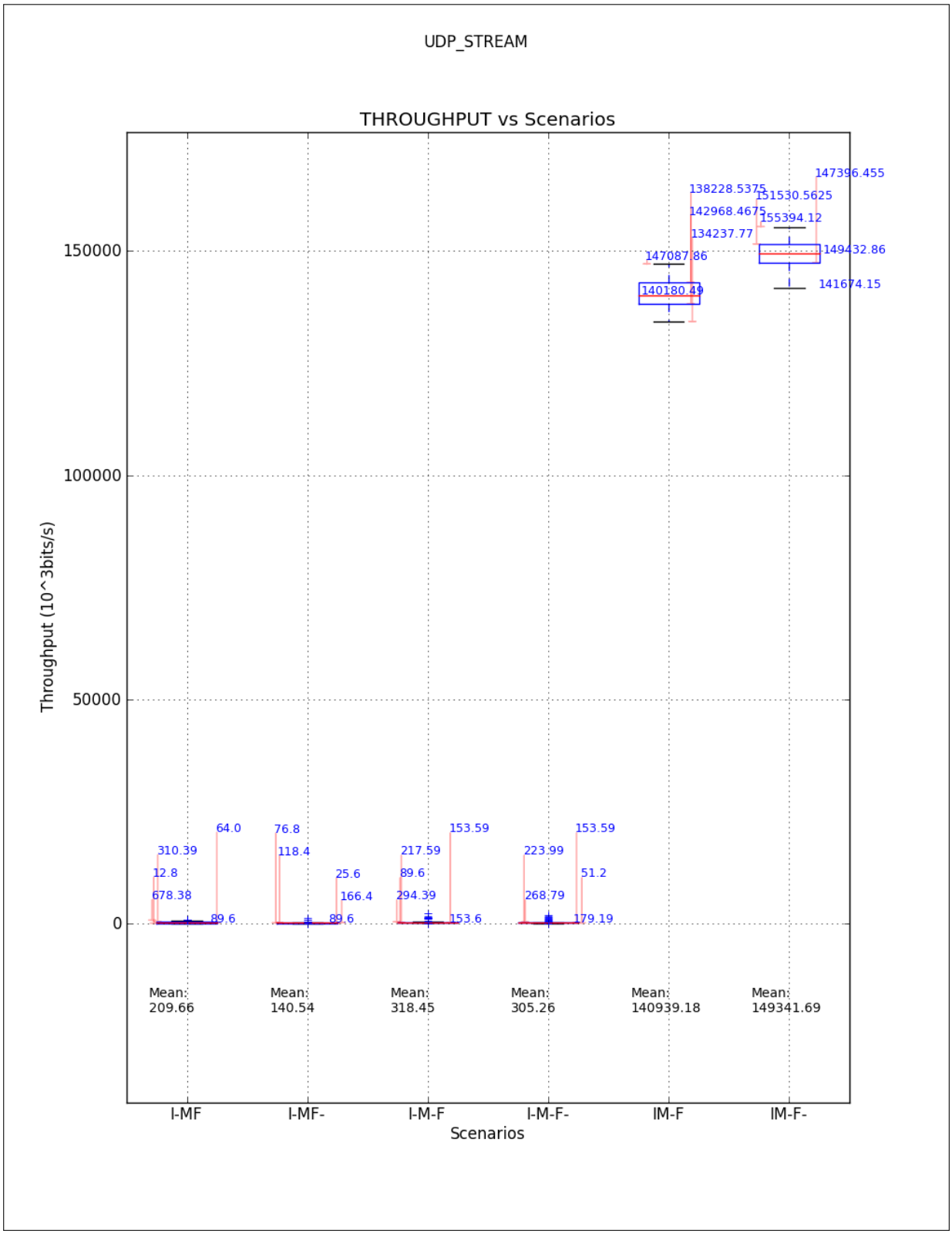

Figure G.1: Boxplot showing the summary of test results for UDP_STREAM test

THIS PAGE INTENTIONALLY LEFT BLANK

REFERENCES

[1] C. E. Irvine, T. D. Nguyen, D. Shifflett, T. E. Levin, J. Khosalim, C. Prince, P. C. Clark, and M. Gondree. MYSEA: The Monterey Security Architecture. In *Proceedings: 2009 ACM workshop on Scalable trusted computing*, pp. 39 – 48, 2009.

[2] C. E. Irvine and T. E. Levin. Quality of security service. In *Proceedings: 2000 workshop on New Security Paradigms*, pp. 91–99, 2000.

[3] C. E. Irvine, D. Shifflett, P. C. Clark, T. E. Levin, and G. Dinolt. Monterey security enhanced architecture project. In *Proceedings: DARPA formation Survivability Conference and Exposition*, volume 2, pp. 176–181, 2003.

[4] C.E. Irvine, T.E. Levin, T.D. Nguyen, D. Shifflett, J. Khosalim, P.C Clark, A. Wong, F. Afinidad, D. Bibighaus, and J. Sears. Overview of a high assurance architecture for distributed multilevel security. In *Proceedings: Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pp. 38–45, 2004.

[5] F. H. John. IPSec-Based Dynamic Security Services for The MYSEA Environment. Master's thesis, Naval Postgraduate School, 2005.

[6] E. Brown. SMTP on a High Assurance Multilevel Server. Master's thesis, Naval Postgraduate School, 2000.

[7] E. Theresa. Enhancement of Internet Message Access Protocol for User-Friendly Multilevel Mail Management. Master's thesis, Naval Postgraduate School, 2000.

[8] L. Tse. Feasibility study of VOIP Integration into the MYSEA Environment. Master's thesis, Naval Postgraduate School, 2005.

[9] J. A. Bradney. Use of WebDAV to support a virtual file system in a coalition environment. Master's thesis, Naval Postgraduate School, 2006.

[10] E. Bersack. Implementation of a HTTP (Web) Server on a High Assurance Multilevel Secure Platform. Master's thesis, Naval Postgraduate School, 2000.

[11] K. L. Ong. Design and Implementation of WIKI services in a multilevel secure environment. Master's thesis, Naval Postgraduate School, 2007.

[12] W.R. Shockley and R. R. Schell. TCB Subsets for incremental evaluation. In *Proceedings: 3rd AIAA Conference on Computer Security*, pp. 131–139, 1987.

[13] D. E. Denning. A lattice model of secure information flow. *Magazine: Communications of the ACM*, 19(5):236–243, 1976.

[14] D. E. Bell and L. J. LaPadula. Secure Computer Systems: Mathematical Foundations. Technical report, MITRE Technical Report 2547, Vol II, May 1973.

[15] J. K. Biba. Integrity Considerations for Secure Computer Systems. Technical report, MITRE Technical Report, April 1977.

[16] L. BAE Systems Information Technology. Security Target, Version 1.11 for XTS-400 Version 6, December 2004. http://www.commoncriteriaportal.org/files/epfiles/ST_VID3012-ST.pdf. Last Accessed: Aug 2012.

[17] Common Criteria. Certification Evaluation Report of BAE Systems STOP OS v7.3.1, Jan 2012. http://www.commoncriteriaportal.org/files/epfiles/383-4-176%20CR%20v1.0e.pdf. Last Accessed: Aug 2012.

[18] KAME. IPSec-Tools, Aug 2012. http://ipsec-tools.sourceforge.net/. Last Accessed: Aug 2012.

[19] J. C. Mogul. Brittle Metrics in Operating Systems Research. In *Proceedings: 7th Workshop on Hot Topics in Operating Systems*, pp. 90 – 95, 1999.

[20] R. H. Saavedra and A. J. Smith. Analysis of Benchmark Characteristics and Benchmark Performance Prediction. *ACM Transactions on Computer Systems*, 14(4):344 – 384, November 1996.

[21] Carl Staelin. lmbench: an extensible micro-benchmark suite. *Software: Practice and Experience*, 35:1079–1105, 2005.

[22] A. B. Brown and M. I. Seltzer. Operating system benchmarking in the wake of LMbench: A case study of the performance of NetBSD on the Intel x86 architecture. In *Proceedings: ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 214–224, 1997.

[23] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, R.C. Whaley, and K. Yelick. Self-Adapting Linear Algebra Algorithms and Software. In *Proceedings: IEEE*, volume 93, pp. 293–312, 2005.

[24] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and performance in the Denali isolation kernel. In *Proceedings: 5th symposium on Operating Systems Design and Implementation*, volume 36, pp. 195–209, 2002.

[25] J. K. Ousterhout. Why Aren't Operating Systems Getting Faster As Fast as Hardware. Technical report, Western Research Laboratory, 1989.

[26] W. Feng., P. Balaji., C. Baron., L. N. Bhuyan, and D.K. Panda. Performance characterization of a 10-Gigabit Ethernet TOE. In *Proceedings: High Performance Interconnects, 2005. Proceedings. 13th Symposium*, pp. 58–63, 2005.

[27] Apache Software Foundation. The Apache Jmeter, Aug 2012. http://jmeter.apache.org/. Last Accessed: Aug 2012.

[28] Netperf. Netperf, 2012. Retrieved Aug 2012. http://www.netperf.org/netperf/. Last Accessed: Aug 2012.

[29] LMBench. LMBench, Aug 2012. http://www.bitmover.com/lmbench/. Last Accessed: Aug 2012.

[30] Netperf. Netperf User Manual and Documentation, 2012. Retrieved Aug 2012. http://www.netperf.org/svn/netperf2/tags/netperf-2.5.0/doc/netperf.html#Installing-Netperf-Bits. Last Accessed: Aug 2012.

[31] V. Paxson. Strategies for Sound Internet Measurement. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pp. 263–271, 2004.

[32] C. Shue, Y. Shin, M. Gupta, and J. Y. Choi. Analysis of ipsec overheads for vpn servers. In *NPSEC'05 Proceedings of the First international conference on Secure network protocols*, pp. 25–30, 2005.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Kris Britton
   National Security Agency
   Fort Meade, MD

4. John Campbell
   National Security Agency
   Fort Meade, MD

5. Deborah Cooper
   DC Associates, LLC
   Reston, VA

6. Louise Davidson
   Institute for Defense Analyses
   Alexandria, VA

7. Rob Dobry
   NSA
   Fort Meade, MD

8. Dr. Steven King
   ODUSD
   Washington, DC

9. Steve LaFountain
   NSA
   Fort Meade, MD

10. Dr. Greg Larson
    IDA
    Alexandria, VA

11. John Mildner
    SPAWAR
    Charleston, SC

12. Ed Schneider
    IDA
    Alexandria, VA

13. Dr. Cynthia E.Irvine
    Naval Postgraduate School
    Monterey, CA

14. Dr. Mark Gondree
    Naval Postgraduate School
    Monterey, CA

15. Professor Yeo Tat Soon
    Temasek Defence Systems Institute National University of Singapore
    Singapore

16. Tan Lai Poh
    Temasek Defence Systems Institute National University of Singapore
    Singapore

17. Mr Teo Tiat Leng
    Defence Science Technology and Agency of Singapore
    Singapore

18. My Chua Kai Ping
    Defence Science Technology and Agency of Singapore
    Singapore