IEEE SecDev | 2020

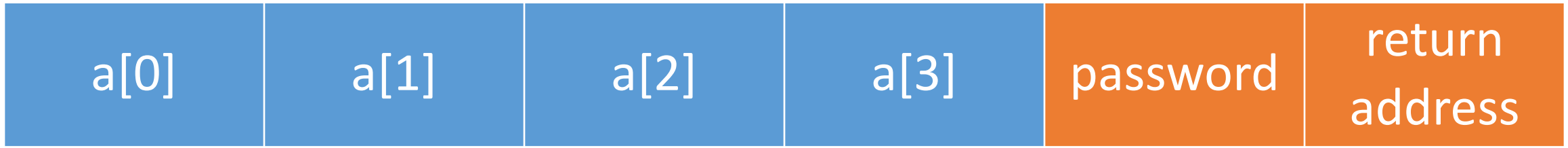
IEEE COMPUTER SOCIETY

Refactoring the FreeBSD Kernel with Checked C

Junhan Duan, Yudi Yang, Jie Zhou, and John Criswell

UNIVERSITY of ROCHESTER

C Lacks Memory Safety!



What if:

1. a[4] ← 10
2. a[5] ← malicious function

Buffer Overflow Attack

Checked C: Spatial Memory Safety for C

- Compatible with existing C code
- Designed for incremental change

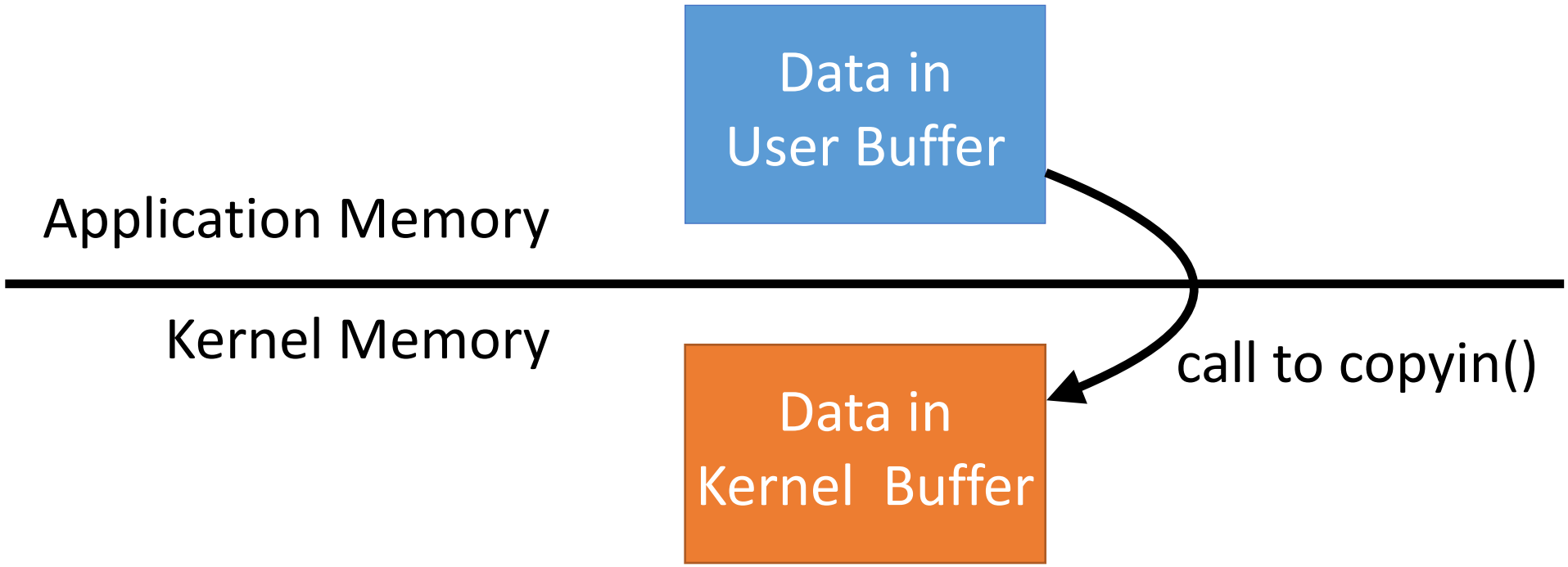
Checked C: Spatial Memory Safety for C

- New pointer types: ptr<T> and array_ptr<T>
- Bounds Safe Interface
  - Allow checked and unchecked code to inter-operate
  - Polymorphic pointer types for function arguments
- Checked Region
  - All code within region guaranteed to be spatially safe
  - Code within are restricted to use checked pointers only

Our Refactoring Effort

- Kernel Support Routine: Memory manipulations
  - memcpy(), memmove(), memset()
  - malloc(), free()
- System Calls
  - Calls to copyin(), copyout(), copyinstr()
- TCP/IP Stack
  - mbuf code, IP and UDP packet manipulation code

Copying Data Between Applications and Kernel



TCP / IP Stack

- mbuf package
    - network packet manipulation
- ip_input() & ip_output() & relevant functions
    - process ip headers
- udp_input() & udp_output() & relevant functions
    - process udp headers

Reflections on Checked C

- **Benefits**
  - Compatible
  - Incremental
- **Limitations**
  - Compiler: p + x - x ≠ p
  - Macro arguments cannot take both checked and unchecked pointers

Reflections on Checked C

Compatible & **Incremental**

- Supports C pointer types
- Bounds Safe Interface
  - Inter-operation between checked and unchecked code
- Checked Region
  - Ensure spatial memory safety

Reflections on Checked C

- **Limitations**
  - Compiler: lacking support for inferring expression equivalence
    - p + x - x ≠ p
  - Macros cannot take both checked and unchecked pointers
  - Casting cannot be used as function inputs

Inconvenience with Macros

Original

- #define memcpy …

Modified

- #define memcpy …
- #define memcpy_checked …

Evaluation

- Refactoring Effort
- Performance Overhead
  - System Calls
  - UDP/IP
- Code Size Overhead

Evaluation - Refactoring Effort

Added	Deleted	Files Modified
1,779	587	61

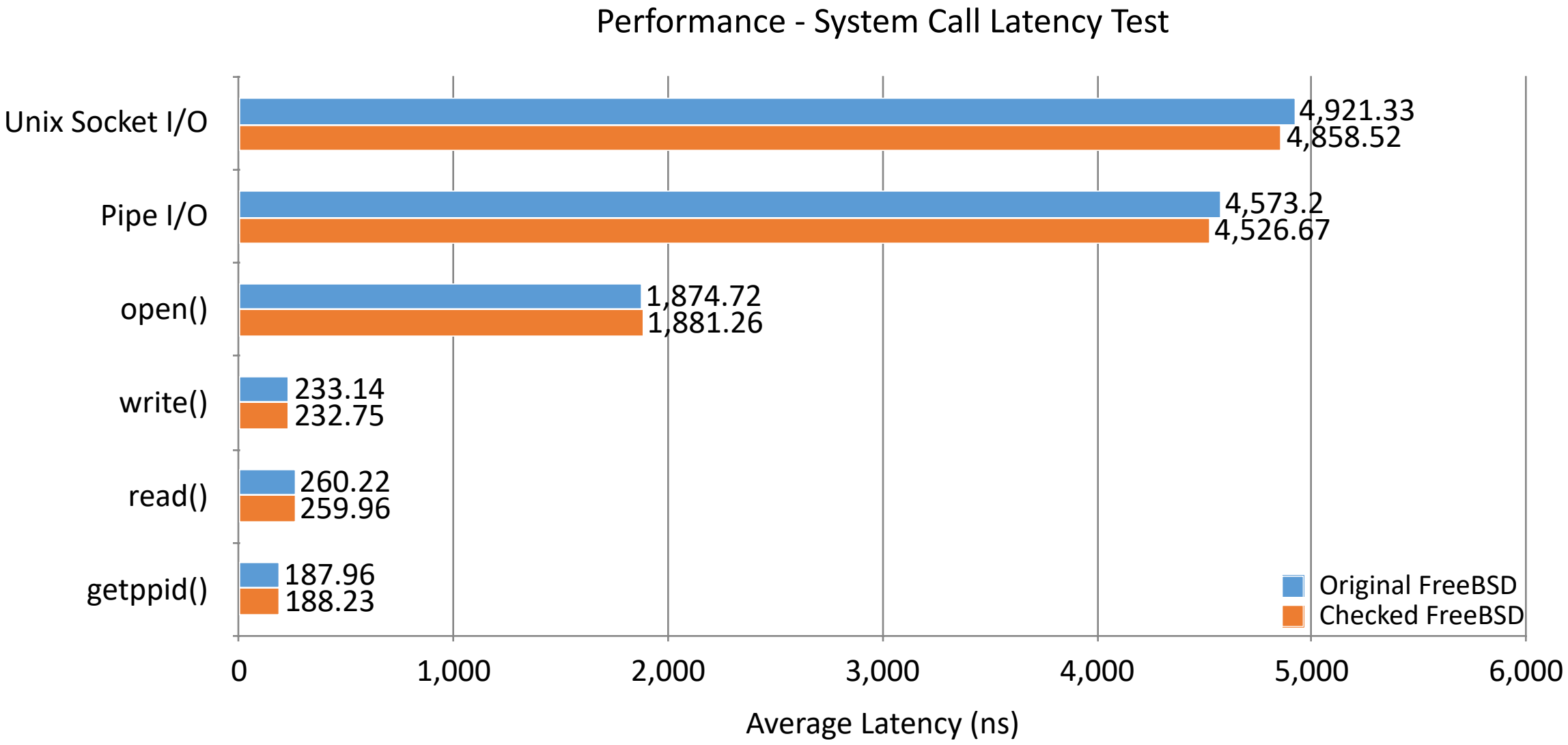
Lines of code modified on FreeBSD kernel

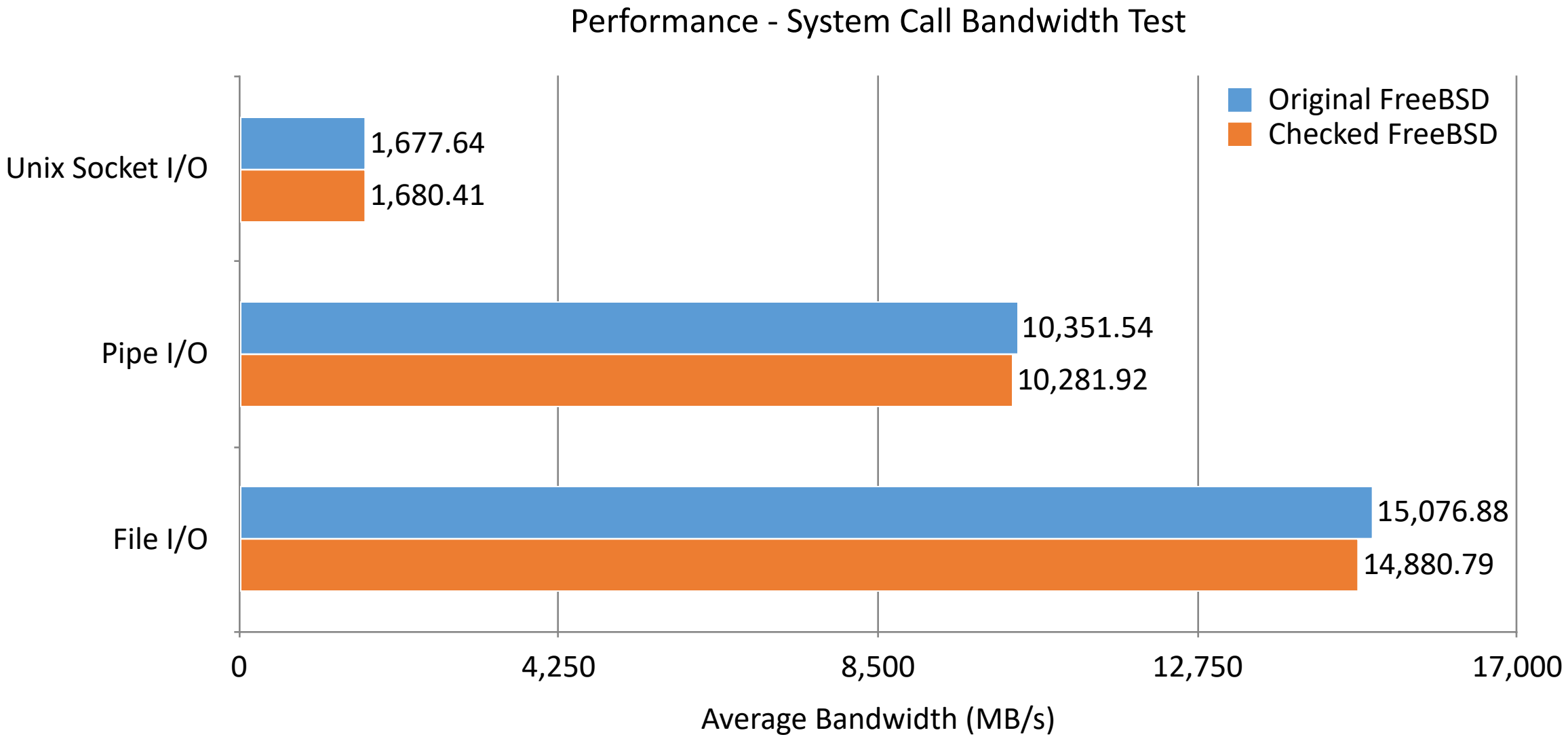
Evaluation - Refactoring Effort

- Refactored by two undergraduate students
- Spent 3 months, 7.3 hours a week
- Around 3 hours a week spent on implementation for the project

Evaluation - Part I

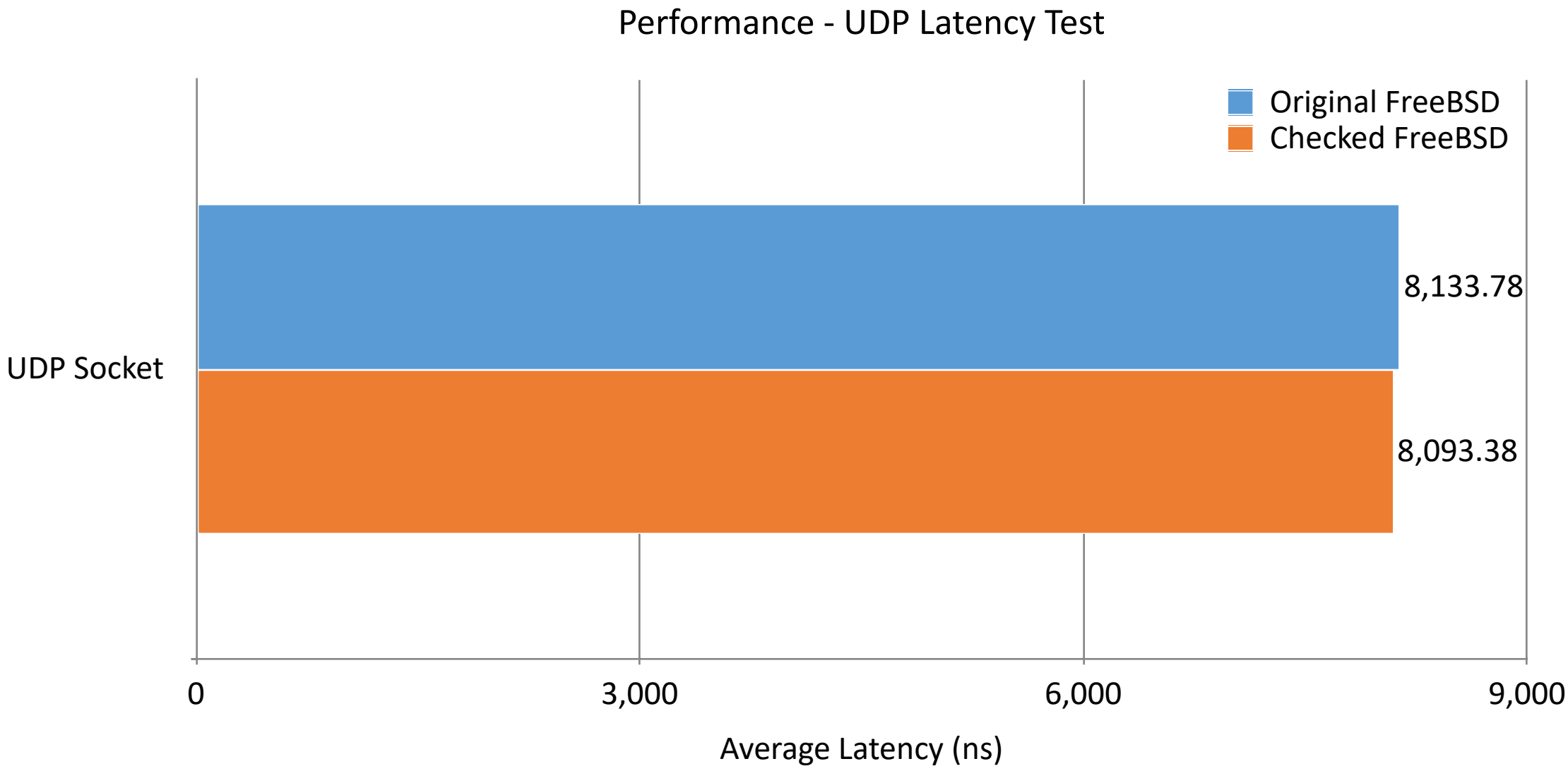
- System calls performance: tested using LMBench suite
- Test of subsystems
  - File System: the bandwidth for file read
  - System Calls: the latency of common system calls
  - Pipe: the bandwidth and transaction latency of pipe I/O
  - Unix Socket: the bandwidth and transaction latency for Unix domain sockets
- Test of system calls:
  - copyin() and copyout(): read(), write(), send(), recv()
  - copyinstr(): open()

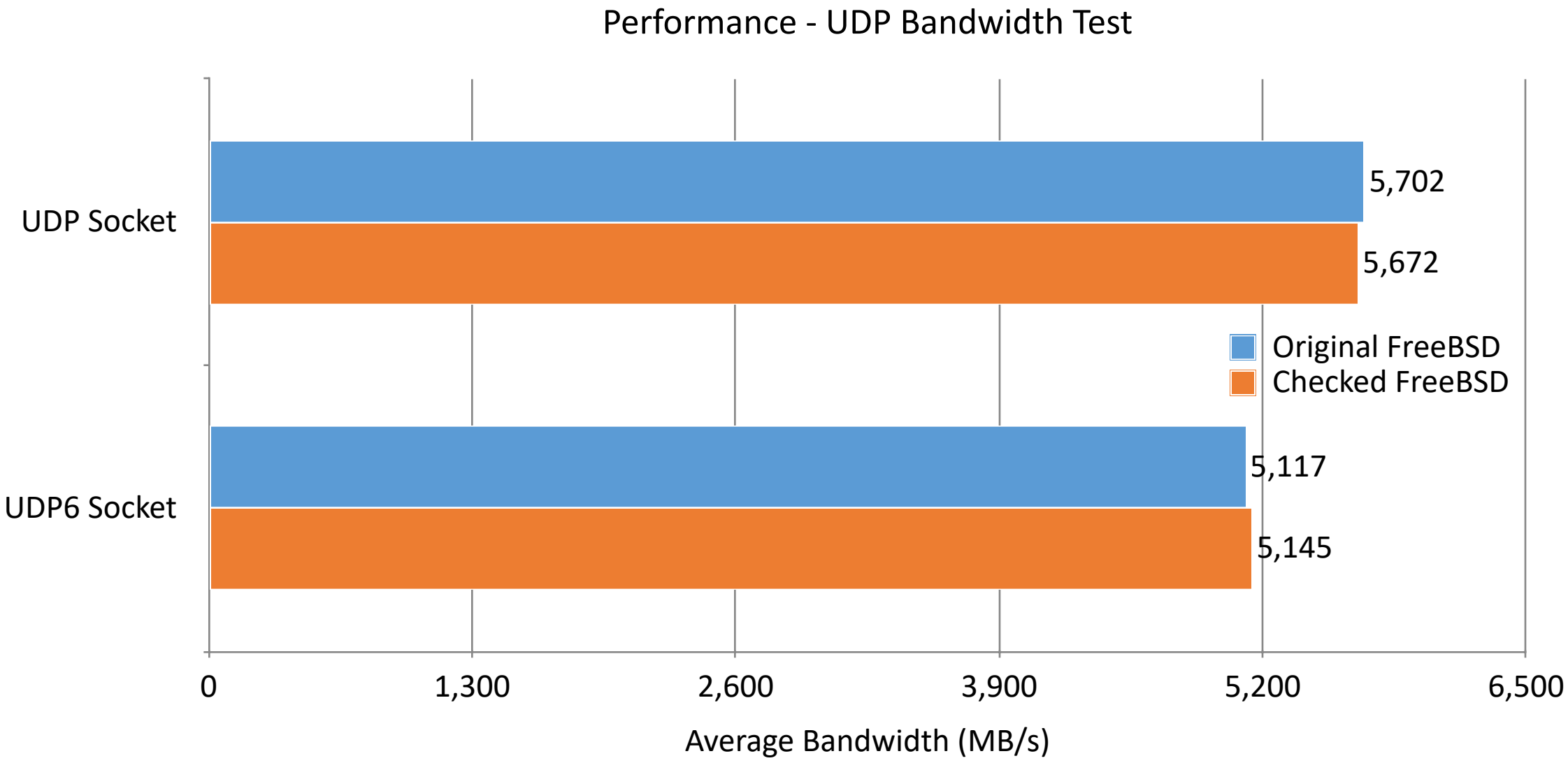

Performance - System Call Latency Test
Unix Socket I/O
Pipe I/O
open()
write()
read()
getppid()
4,921.33
4,858.52
4,573.2
4,526.67
1,874.72
1,881.26
233.14
232.75
260.22
259.96
187.96
188.23
Original FreeBSD
Checked FreeBSD
0
1,000
2,000
3,000
4,000
5,000
6,000
Average Latency (ns)


Performance - System Call Bandwidth Test
Original FreeBSD
Checked FreeBSD
Unix Socket I/O
1,677.64
1,680.41
Pipe I/O
10,351.54
10,281.92
File I/O
15,076.88
14,880.79
0
4,250
8,500
12,750
17,000
Average Bandwidth (MB/s)

Evaluation - Part II

- UDP performance:
  - We used LMBench suite and iPerf3 to test UDP performance
  - LMBench: measured UDP latency
  - iPerf3: measured bandwidth for its server process


Performance - UDP Latency Test
Original FreeBSD
Checked FreeBSD
UDP Socket
8,133.78
8,093.38
0
3,000
6,000
9,000
Average Latency (ns)


Performance - UDP Bandwidth Test
UDP Socket
UDP6 Socket
5,702
5,672
5,117
5,145
Original FreeBSD
Checked FreeBSD
0
1,300
2,600
3,900
5,200
6,500
Average Bandwidth (MB/s)

Evaluation - Code Size Overhead

FreeBSD (baseline)	Checked FreeBSD	Checked - Origin
31,295,952	31,295,448	504

Size of kernel in bytes

Conclusion

- Refactored FreeBSD kernel to use checked pointers
    - copyin() and copyout() calls made by system calls
    - Packet manipulation code in IP and UDP
- Spatial memory safety guaranteed for packet manipulation in IP
    - Thwart buffer overflows and buffer overreads
- **Practically no performance or memory overhead**
- Low refactoring cost
    - Usually does not need to refactor the whole program structure/logic
    - Less than 100 hours spent by two undergraduate students to complete this project