Design and Performance of the OpenBSD Stateful Packet Filter (pf)

Daniel Hartmeier

dhartmei@openbsd.org

Systor AG

Introduction

- part of a firewall, working on IP packet level (vs. application level proxies or ethernet level bridges)
- packet filter intercepting each IP packet that passes through the kernel (in and out on each interface), passing or blocking it
- stateless inspection based on the fields of each packet
- stateful filtering keeping track of connections, additional information makes filtering more powerful (sequence number checks) and easier (replies, random client ports)

Motivation

- OpenBSD included IPFilter in the default install
- what appeared to be a BSD license turned out to be non-free
- unlike other license problems discovered by the ongoing license audit, this case couldn't be resolved, IPFilter removed from the tree
- existing alternatives were considered (ipfw), larger code base, kernel dependencies
- rewrite offers additional options, integrates better with existing kernel features

Overview

- Introduction
- Motivation
- Filter rules, skip steps
- State table, trees, lookups, translations (NAT, redirections)
- Benchmarks
- Conclusions

Filter rules

- linear linked list, evaluated top to bottom for each packet (unlike netfilter's chains tree)
- rules contain parameters that match/mismatch a packet
- rules pass or block a packet
- last matching rule wins (except for 'quick', which aborts rule evaluation)
- rules can create state, further state matching packets are passed without rule set evaluation

Skip steps

- transparent optimization of rule set evaluation, improve performance without affecting semantics
- example: ten consecutive rules apply only to packets from source address X, packet has source address Y, first rule evaluated, next nine skipped
- skipping is done on most parameters, in pre-defined order
- parameters like direction (in, out), interface or address family (IPv4/IPv6) partition the rule set a lot, performance increase is significant
- worst case: consecutive rules have no equal parameters, every rule must be evaluated, no additional cost (linked list traversal)

State table

- TCP (sequence number checks on each packet), ICMP error messages match referred to packet (simplifies rules without breaking PMTU etc.)
- UDP, ICMP queries/replies, other protocols, pseudo-connections with timeouts
- adjustable timeouts, pseudo-connections for non-TCP protocols
- binary search tree (AVL, now Red-Black), O(log n) even in worst-case
- key is two address/port pairs

Translations (NAT, redirections)

- translating source addresses: NAT/PAT to one address using proxy ports
- translating destination: redirections (based on addresses/ports)
- mapping stored in state table
- application level proxies (ftp) in userland

State table keys

- one state entry per connection, stored in two trees
- example: 10.1.1.1:20000 -> 62.65.145.30:50001 -> 129.128.5.191:80
- outgoing packets: 10.1.1.1:20000 -> 129.128.5.191:80, replace source address/port with gateway
- incoming packets: 129.128.5.191:80 -> 62.65.145.30:50001, replace destination address/port with local host
- three address/port pairs of one connection: lan, gwy, ext
- without translation, two pairs are equal

State table keys

- two trees: tree-lan-ext (outgoing) and tree-ext-gwy (incoming), contain the same state pointers
- no addition translation map (and lookup) needed

Normalization

- IP normalization (scrubbing) to remove interpretation ambiguities, like overlapping fragments (confusing IDSs)
- reassembly (caching) of fragments before filtering, only complete packets are filtered
- sequence number modulation

Logging

- through bpf, virtual network interface pflog0
- link layer header used for pf related information (rule, action)
- binary log files, readable with tcpdump and other tools

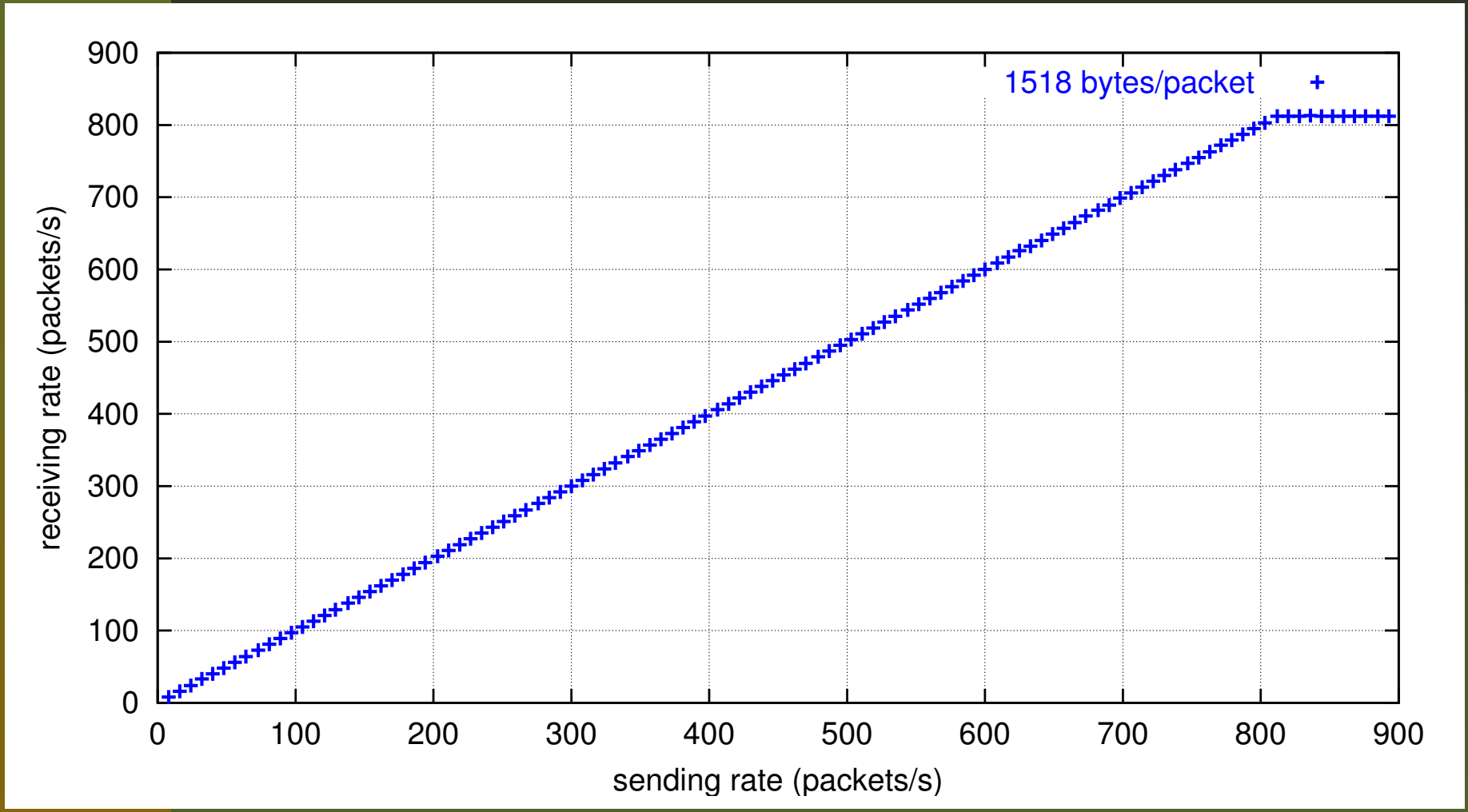
Benchmarks: Setup

- two (old) i386 machines with two network interface cards each, connected with two crosswire Cat5 cables, 10 mbit/s unidirectional
- tester: generate TCP packets on ethernet level through first NIC, capture incoming ethernet frames on second NIC
- firewall: OpenBSD and GNU/Linux (equal hardware), IP forwarding enabled, packet filter enabled, no other services, no other network traffic (static arp table)

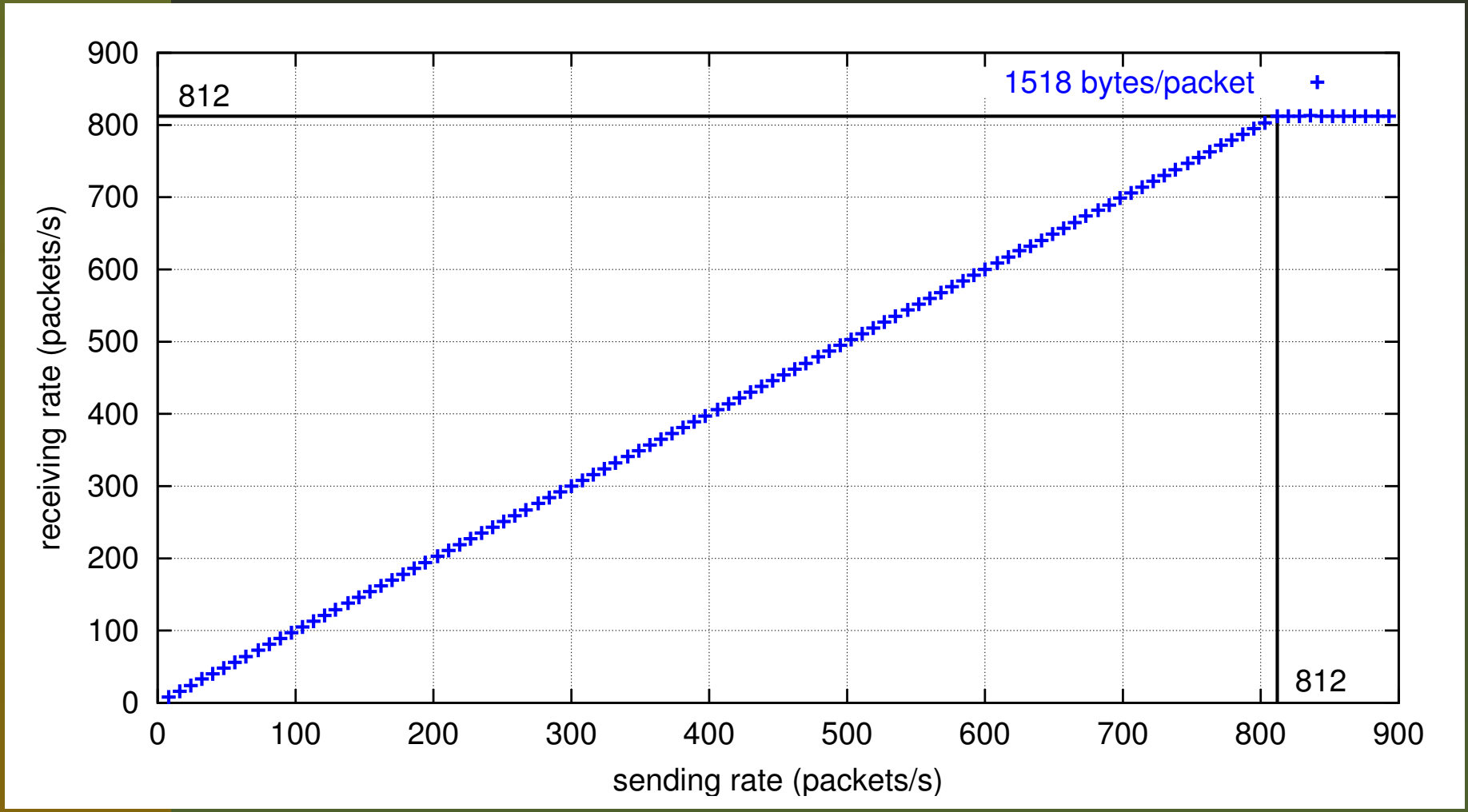
Benchmarks: Packet generation

- TCP packets of variable size, random source/destination addresses and ports
- embedded timestamp to calculate latency, incremental serial number to detect packet loss
- send packets of specified size at specified rate for several seconds, print throughput, latency and loss
- verify that setup can handle maximum link rate correctly

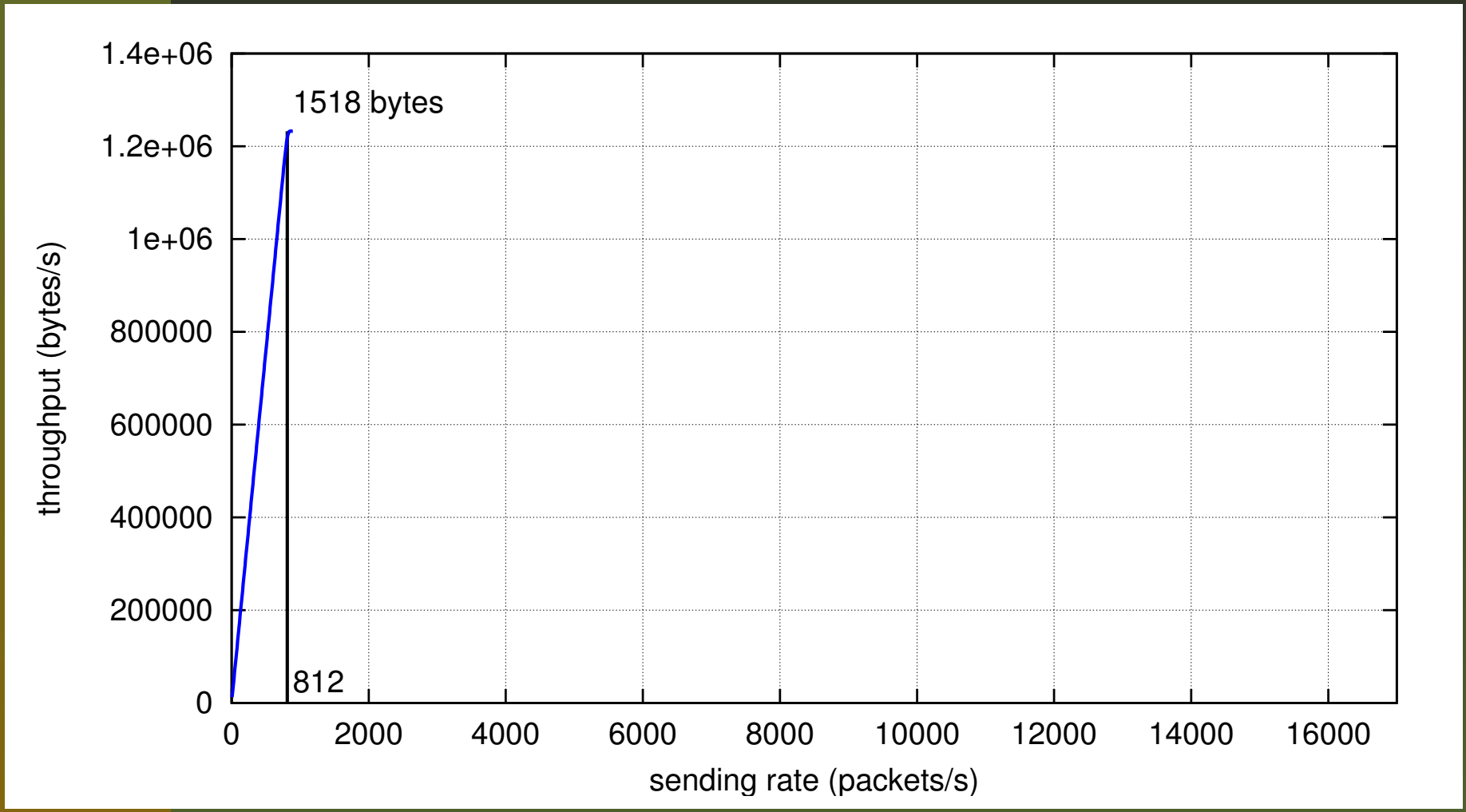
Local, reaching link limit



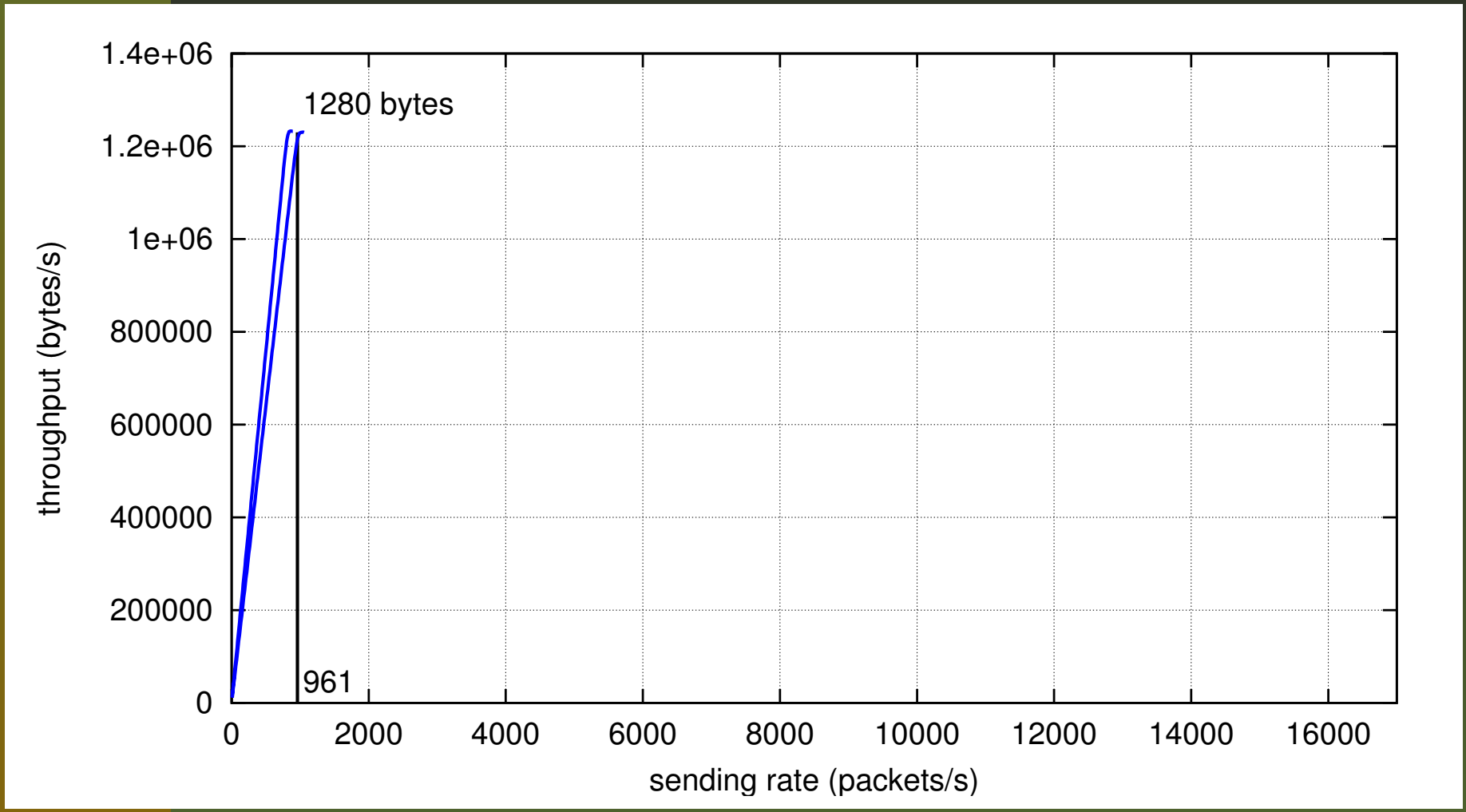
Local, reaching link limit



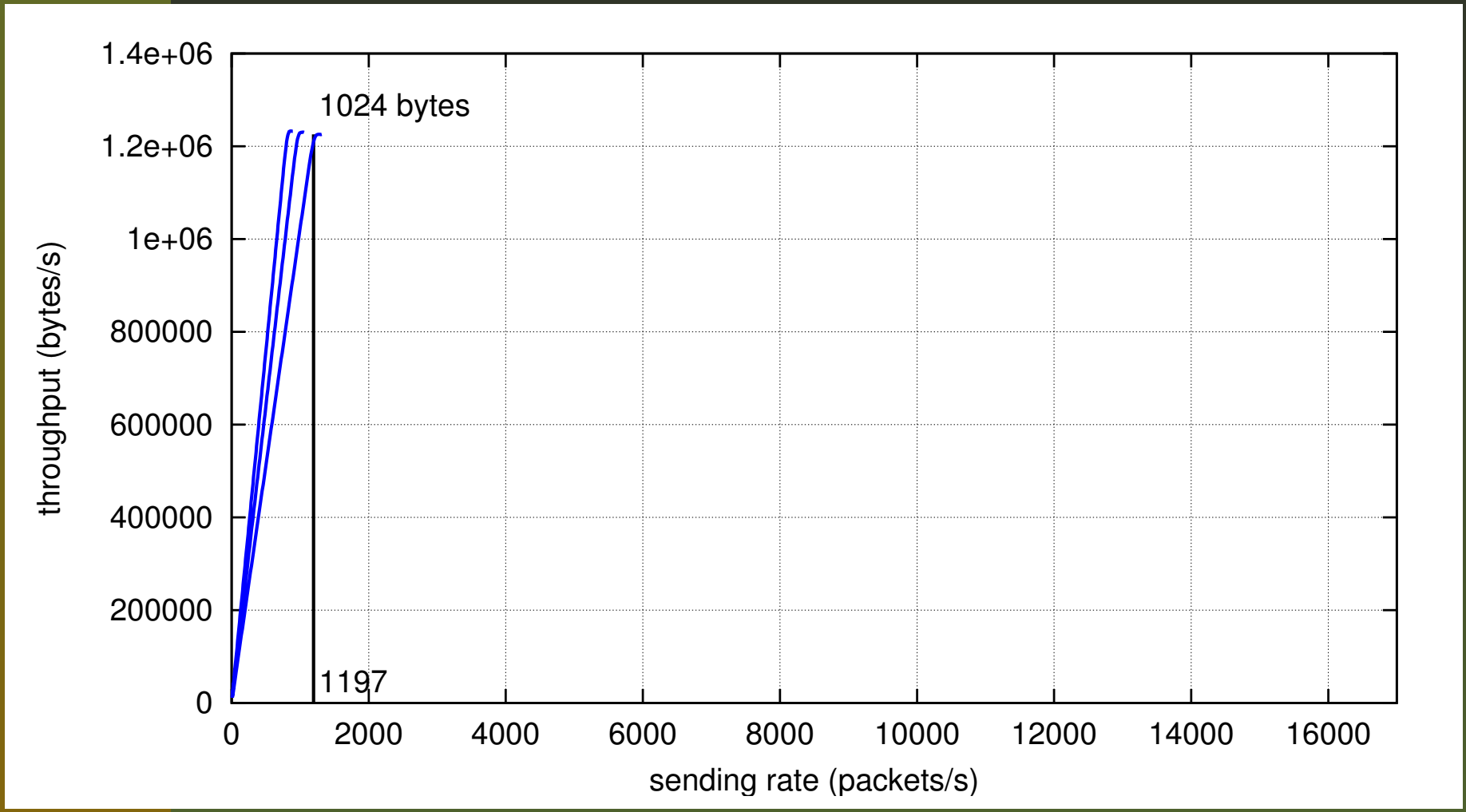
Local, varying packet sizes



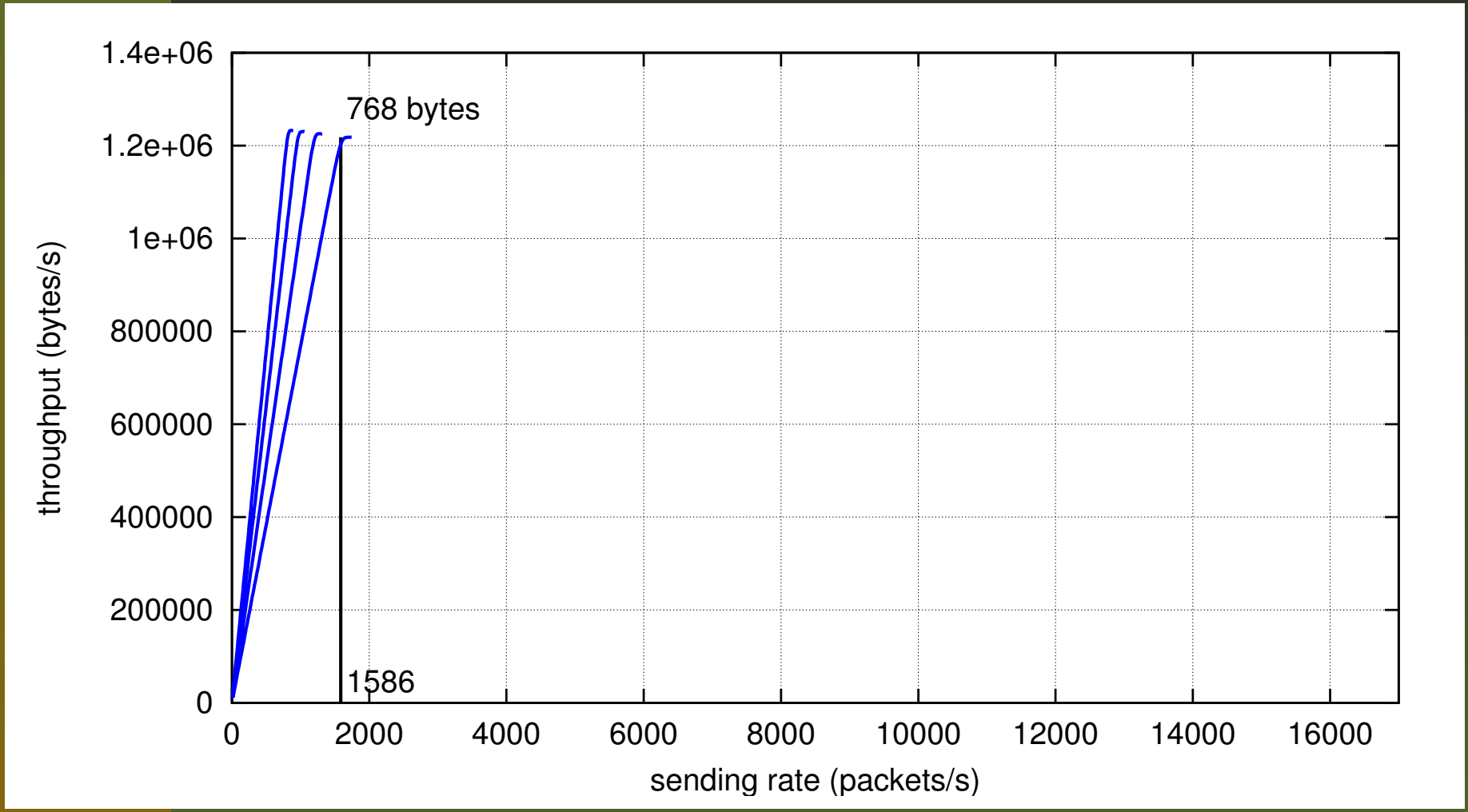
Local, varying packet sizes



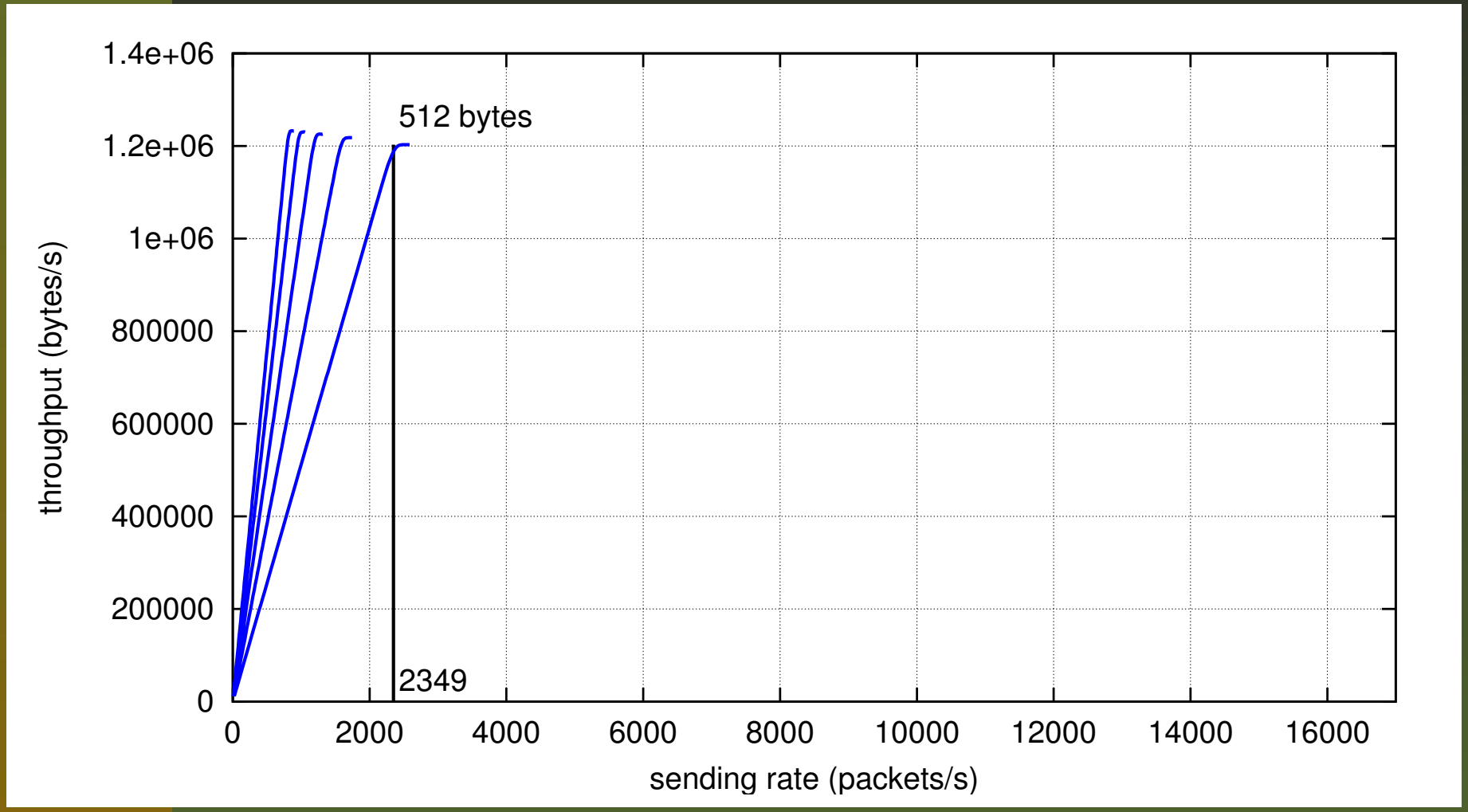
Local, varying packet sizes



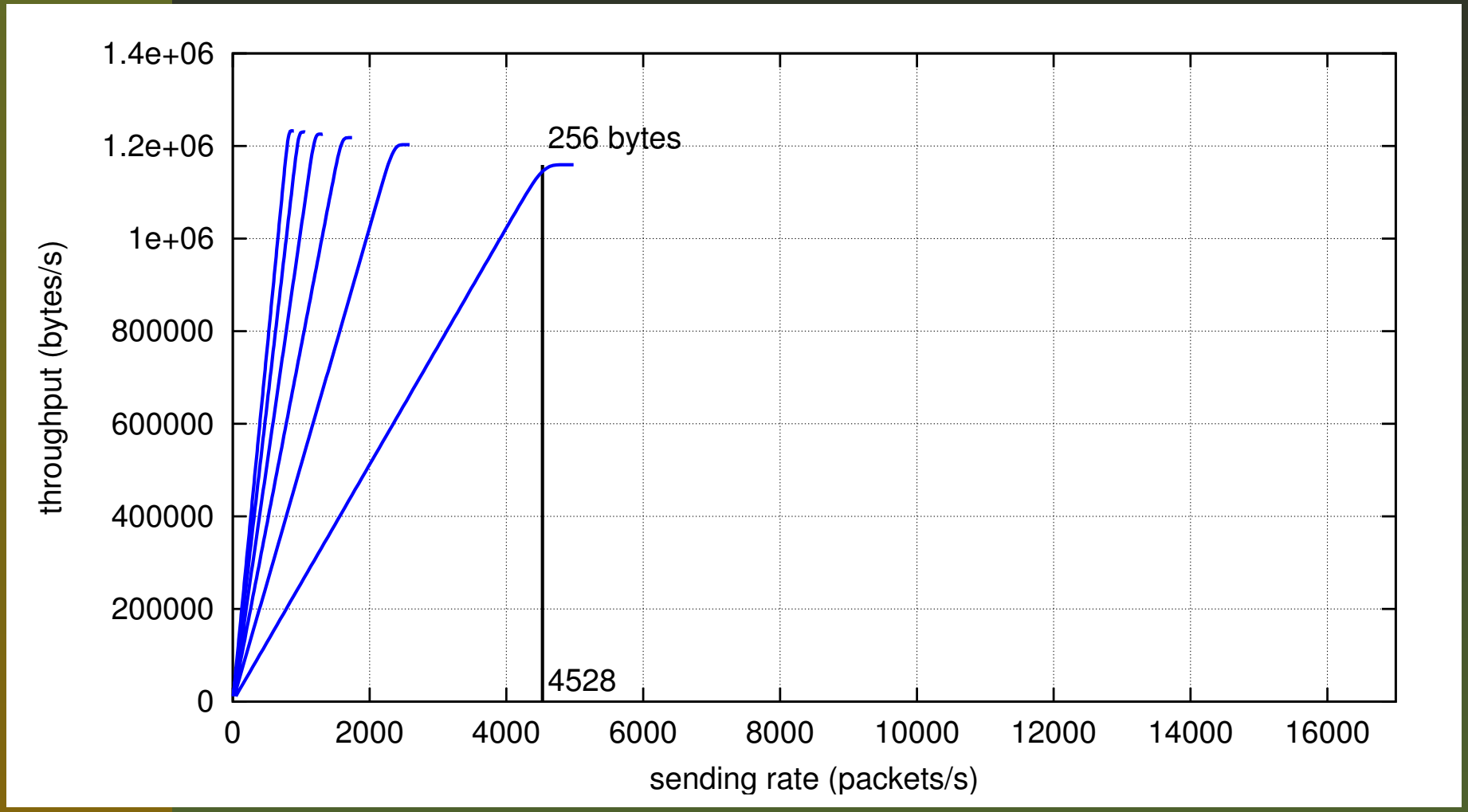
Local, varying packet sizes



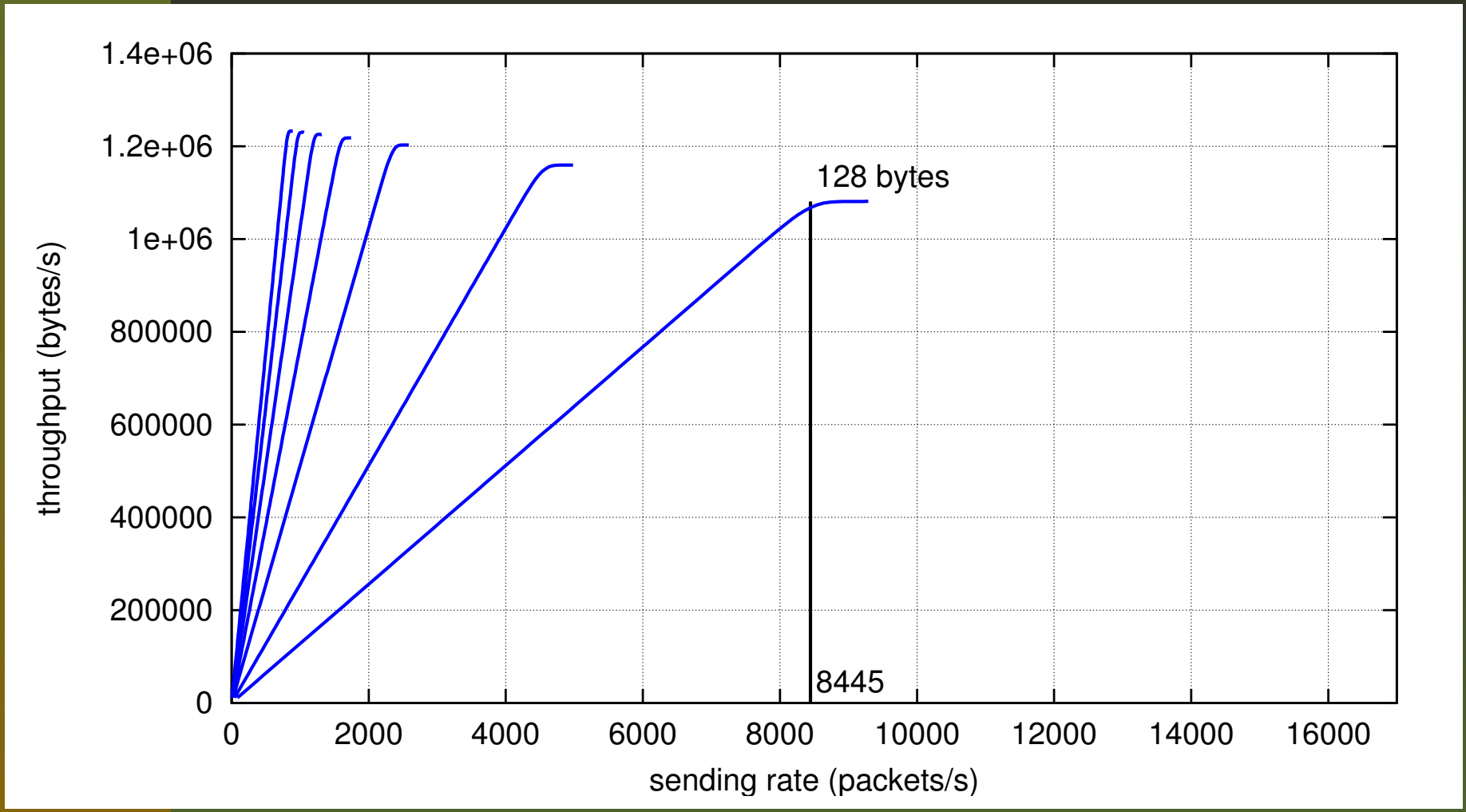
Local, varying packet sizes



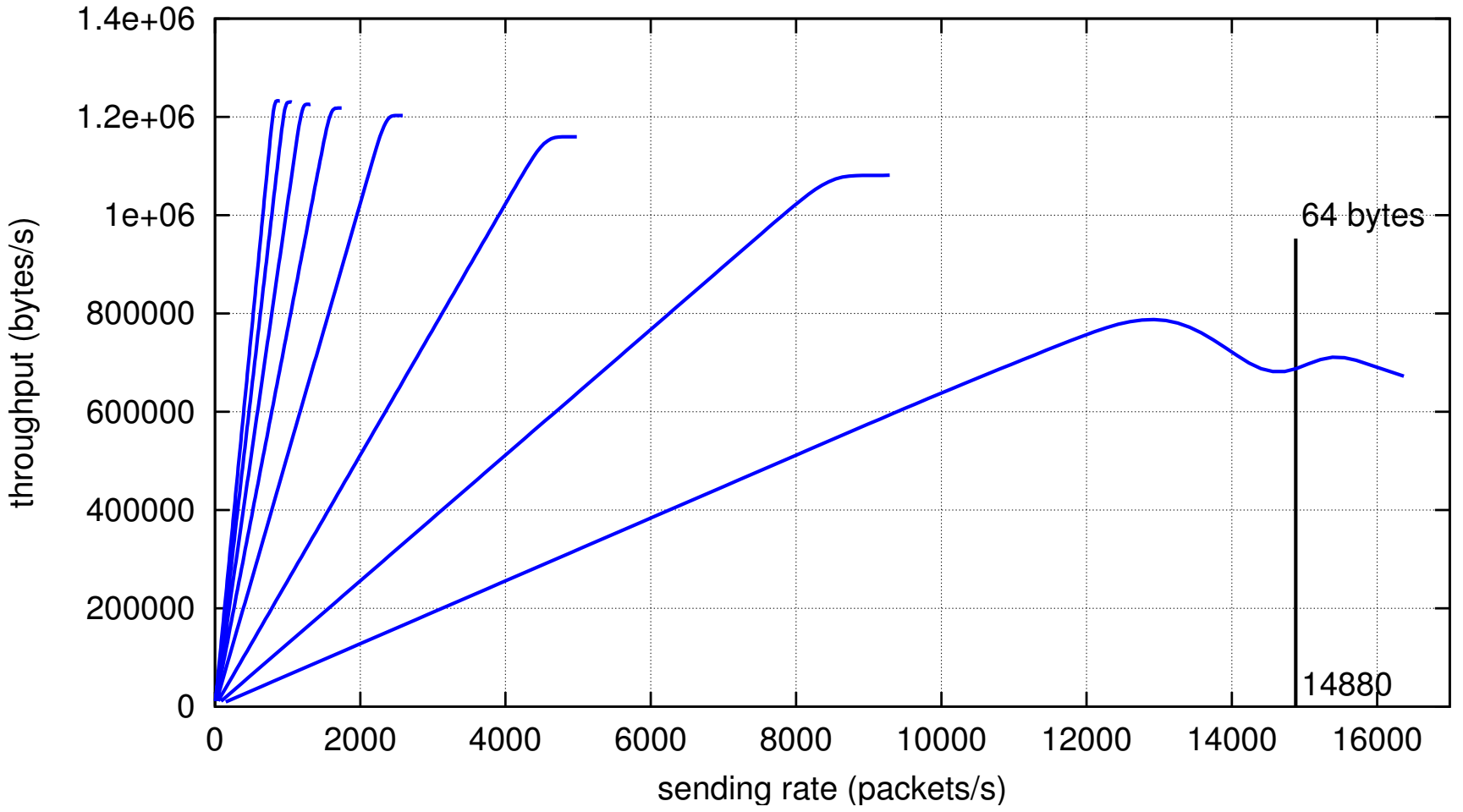
Local, varying packet sizes



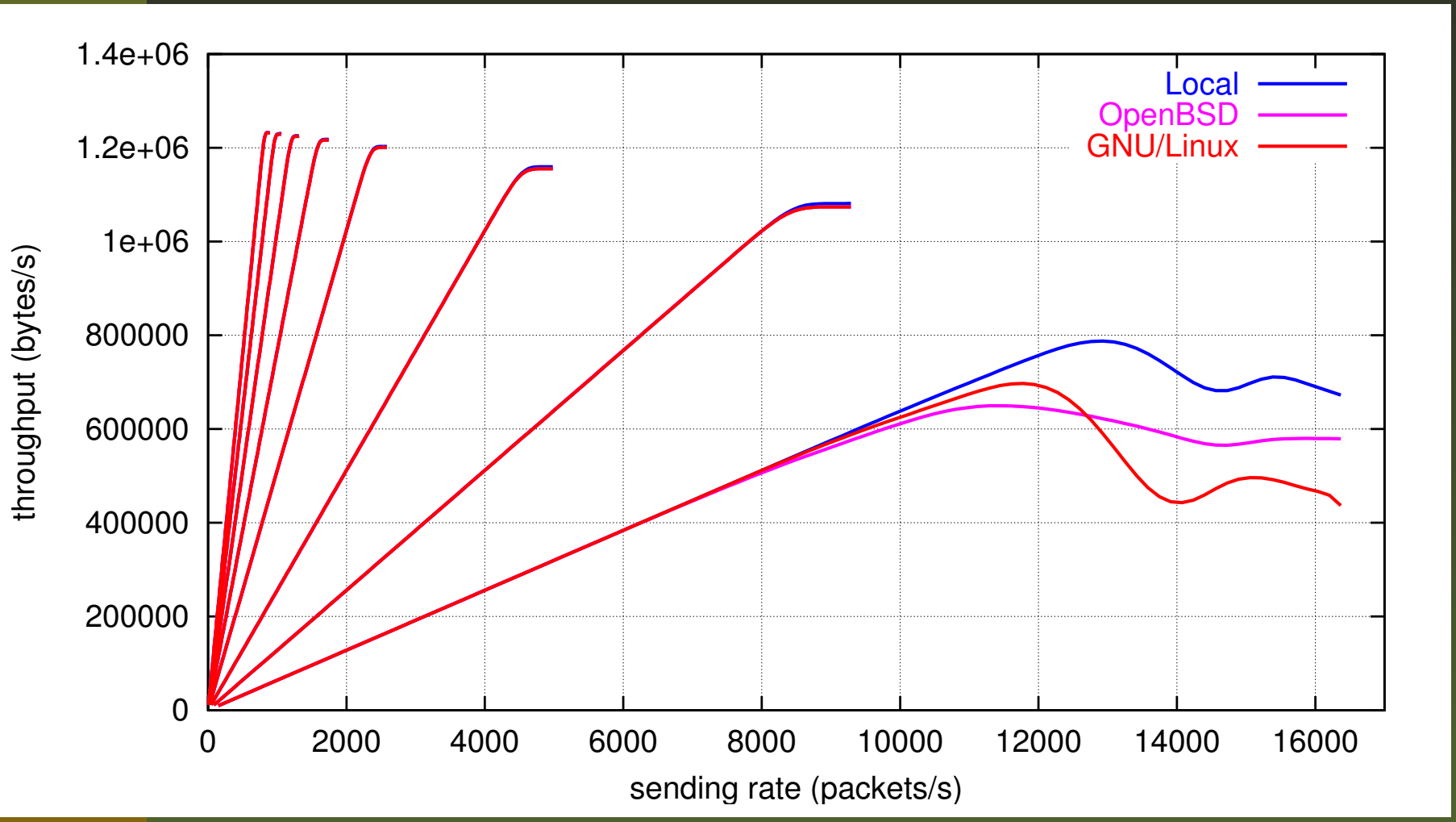
Local, varying packet sizes



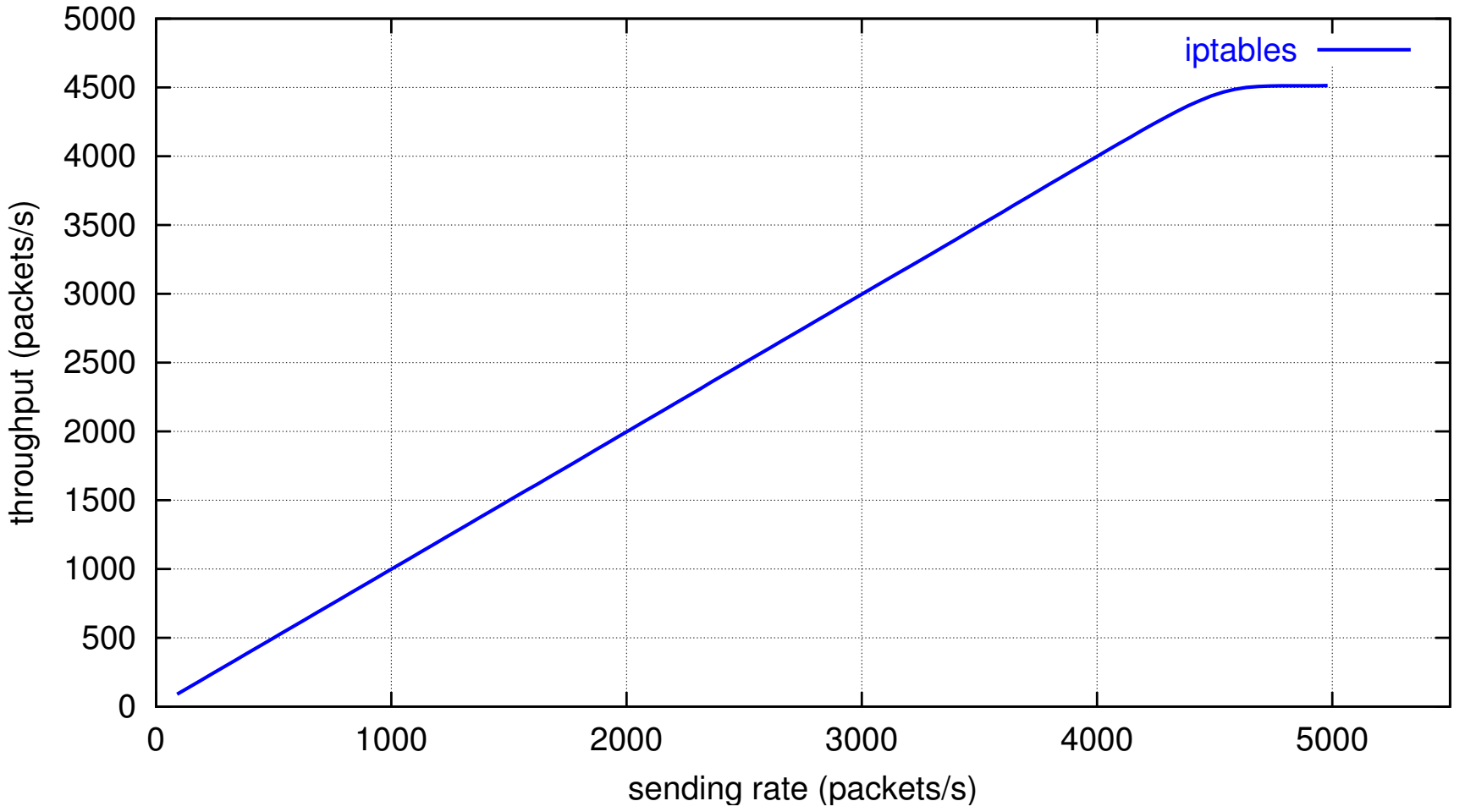
Local, varying packet sizes



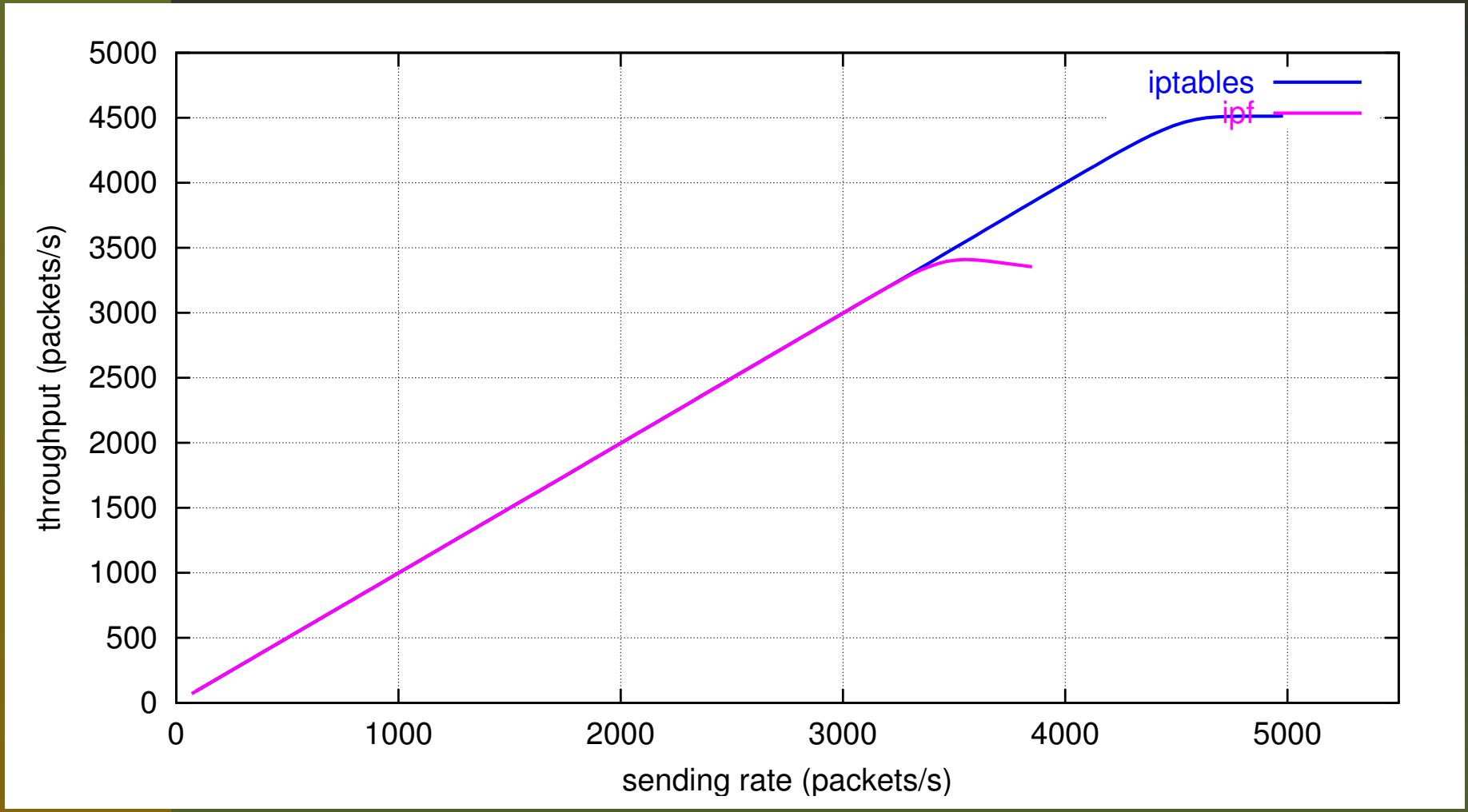
Local, varying packet sizes



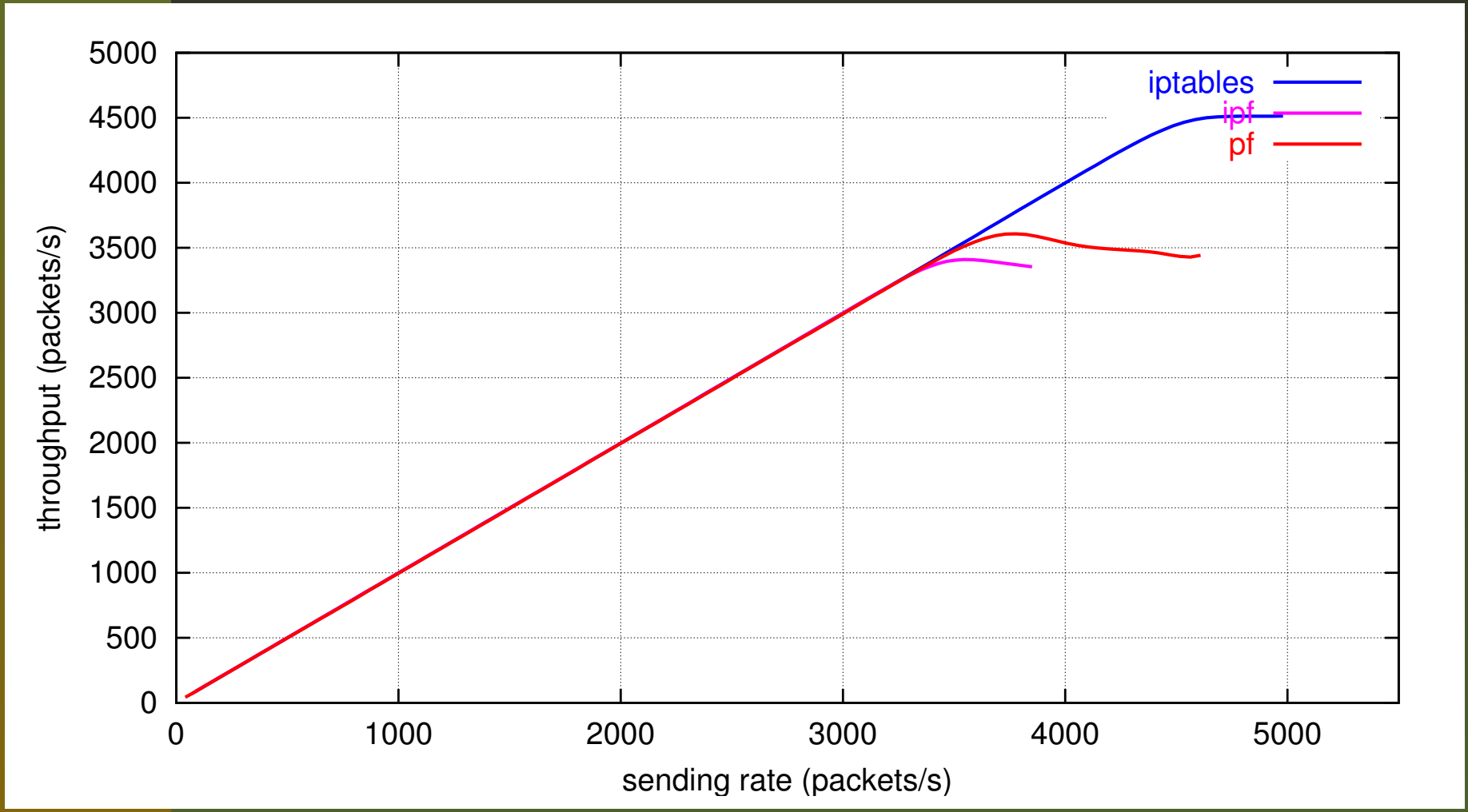
Stateless, 100 rules, throughput



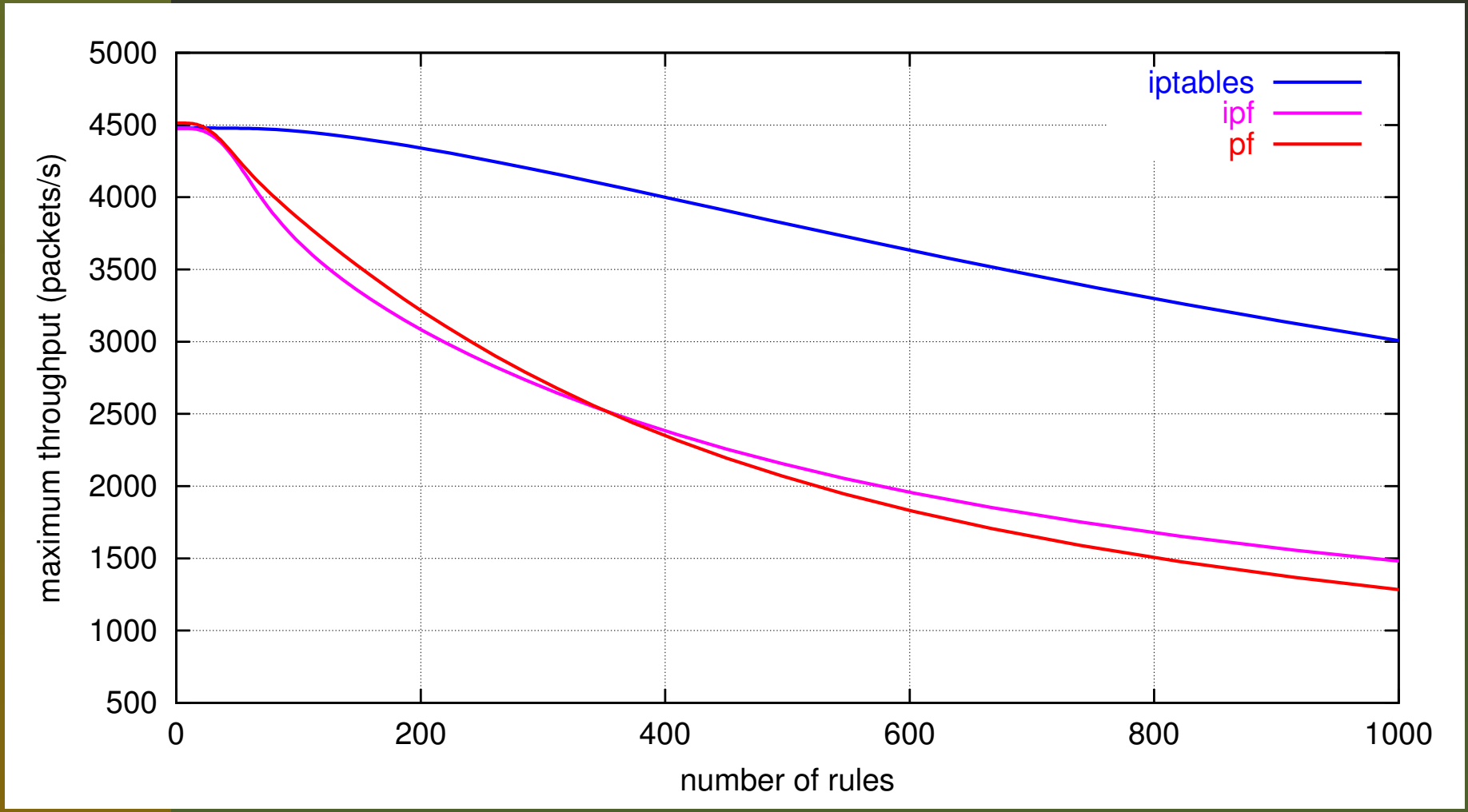
Stateless, 100 rules, throughput



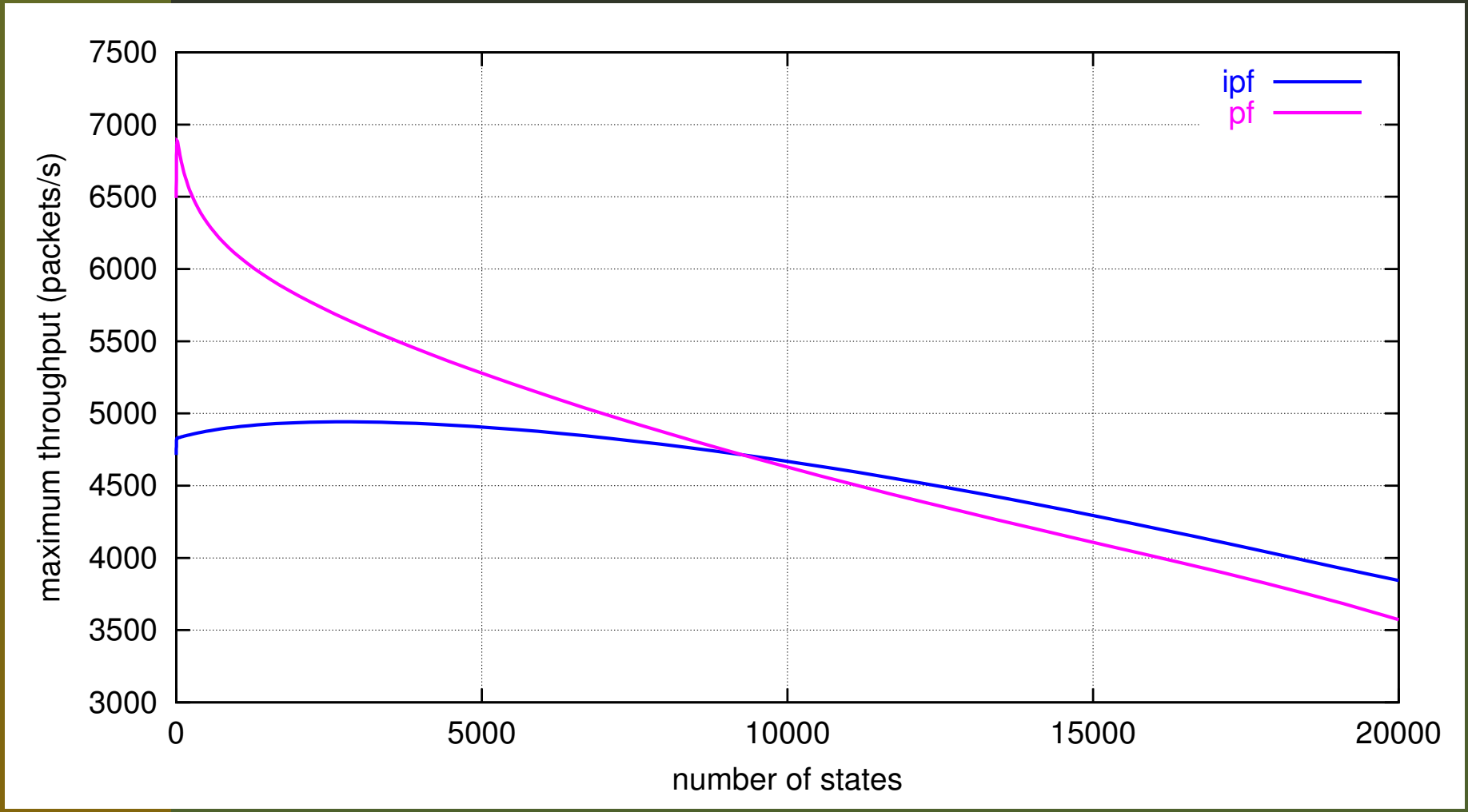
Stateless, 100 rules, throughput



Maximum throughput vs. rules



Maximum throughput vs. states



Conclusions

- rule set evaluation is expensive. State lookups are cheap
- filtering statefully not only improves filter decision quality, it actually increases performance
- memory cost: 64000 states with 64MB RAM (without tuning), increasing linearly
- binary search tree for states scales with O(log n)

Production results

- Duron 700MHz, 128MB RAM, 3x DEC 21143 NICs
- 25000-40000 concurrent states
- average of 5000 packets/s
- fully stateful filtering (no stateless passing)
- CPU load doesn't exceed 10 percent
- (same box and filter policy with IPFilter was 90 percent load average)

Questions?

- The OpenBSD Project: http://www.openbsd.org/
- Paper and slides: http://www.benzedrine.cx/pf.html
- dhartmei@openbsd.org