Why protocol stacks should be in user-space?

Michio Honda (NEC Europe),

Joao Taveira Araujo (UCL),

Luigi Rizzo (Universitea de Pisa),

Costin Raiciu (Universitea Buchalest)

Felipe Huici (NEC Europe)

IETF87 MPTCP WG July 30, 2013 Berlin

Motivation

- Extending layer 4 functionality addresses a lot of problems
  - Increased performance
    - MPTCP, WindowScale, FastOpen, TLP, PRR
  - Ubiquitous encryption
    - TcpCrypt

Is it really possible to deploy layer 4 extensions?

- Networks still accommodate TCP extensions
  - 86 % of the paths are usable for well-designed TCP extensions

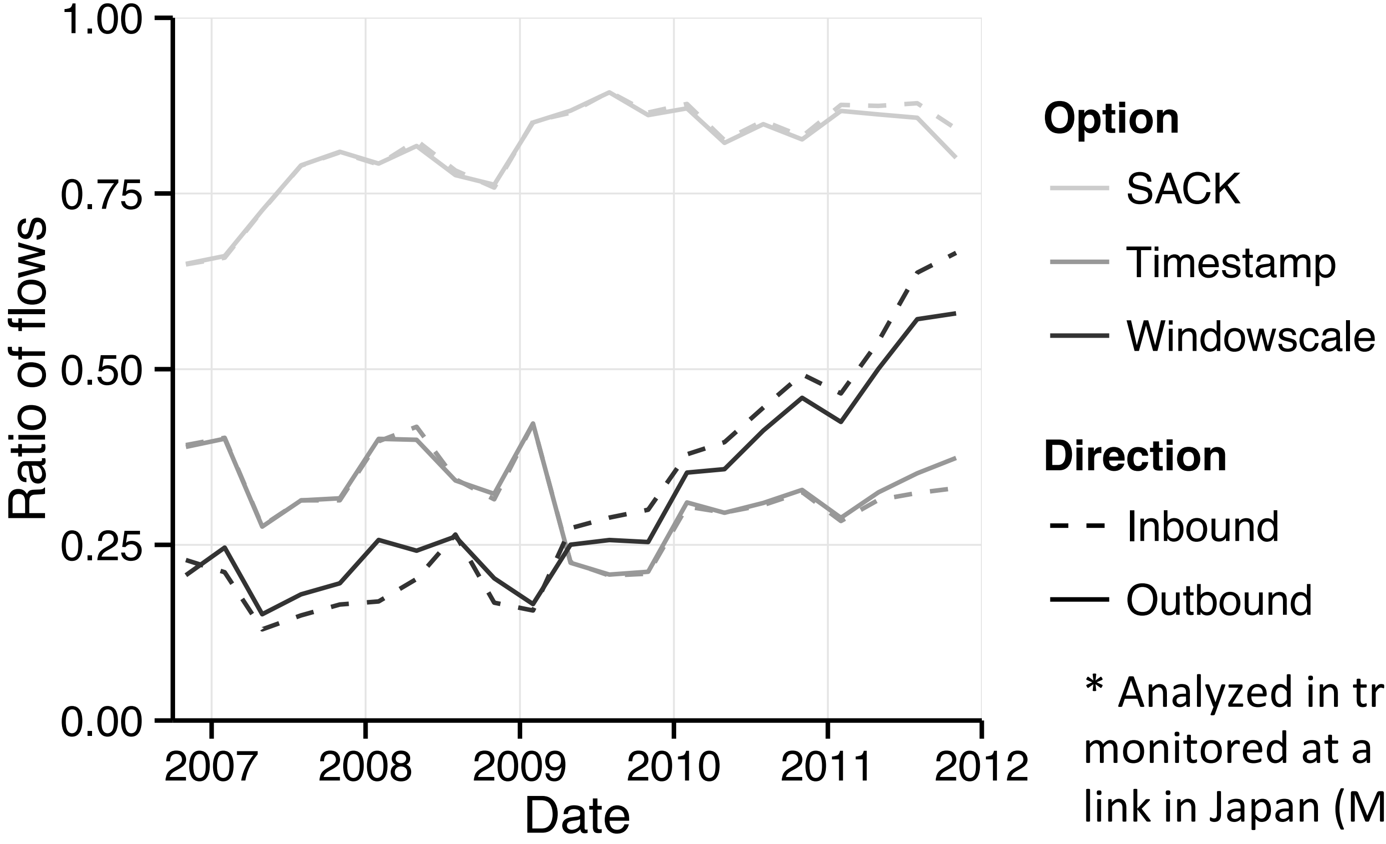
Protocol stacks in end systems

- OSes implement stacks
  - high performance
  - Isolation between applications
  - Socket APIs
- New OS versions adopt new protocols/extensions

Extending protocol stacks: reality

- OSes' release cycle is slow
- Support in the newest OS version does not imply deployment
  - Stakeholders are reluctant to upgrade their OS
  - Disabling a new feature by default also hinders timely deployment

How long does deployment take?

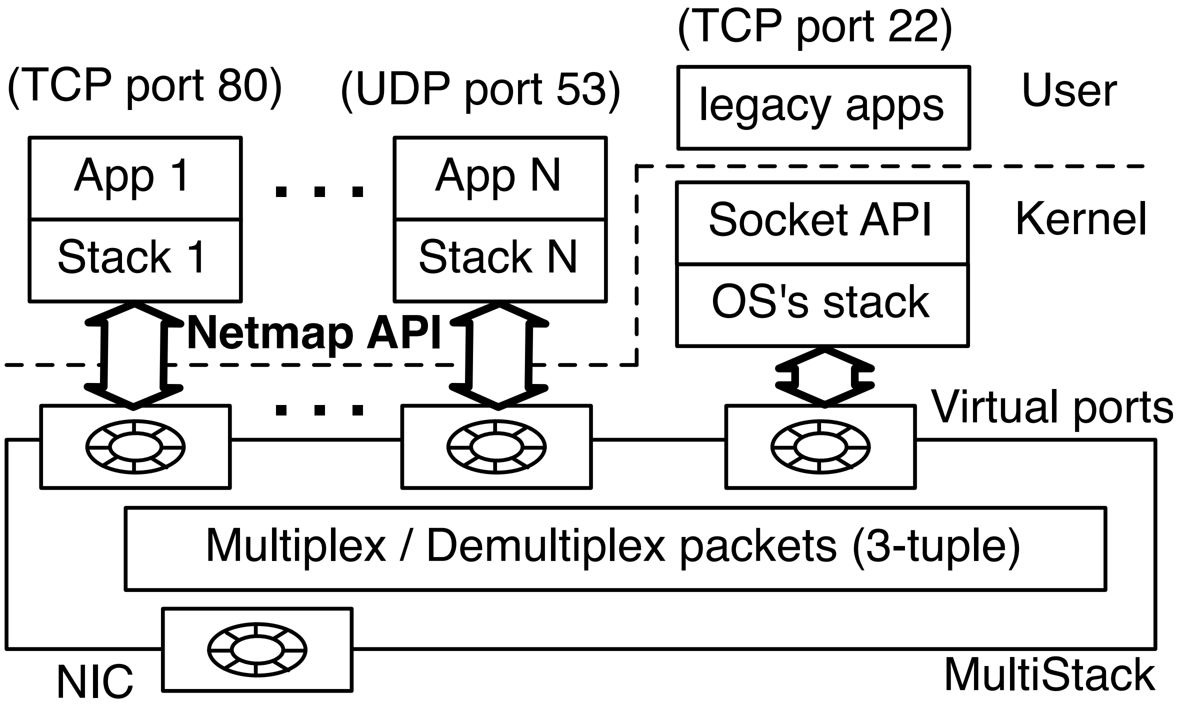


* Analyzed in traffic traces monitored at a single transit link in Japan (MAWI traces)

- SACK is default since Windows 2000
- WS and TS are implemented in Windows 2000, but enabled as default since Windows Vista

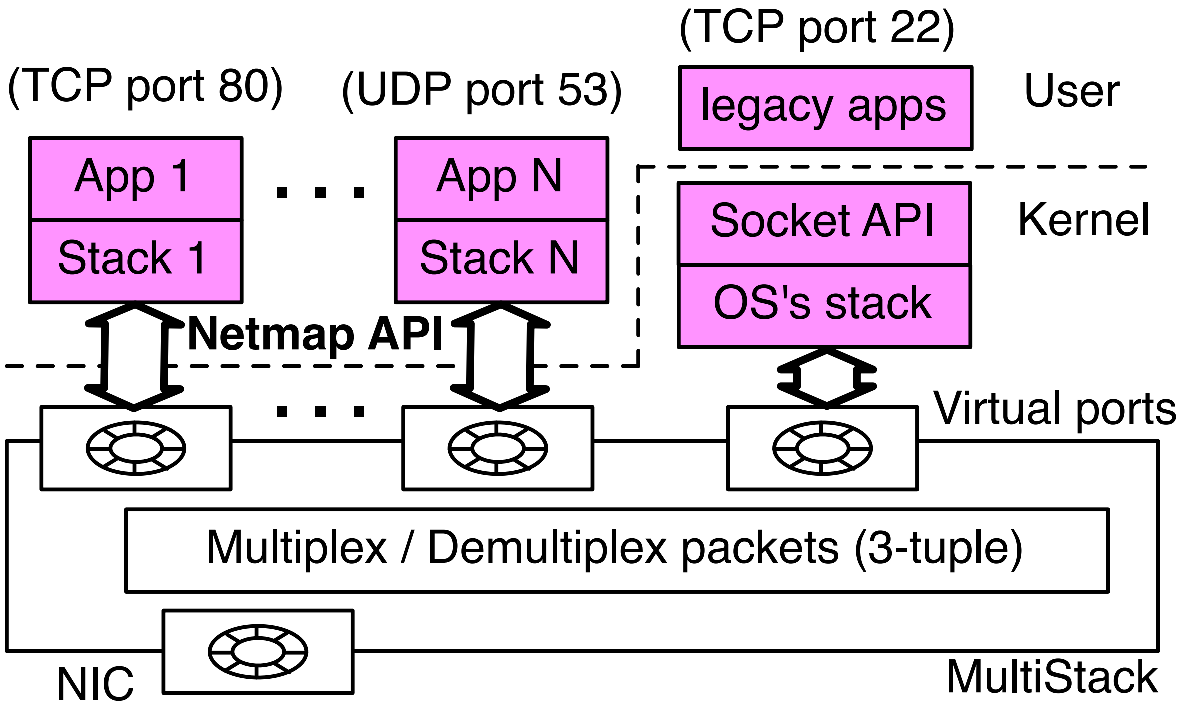
- To ease upgrade, we need to **move protocol stacks up into user-space**
- **Problem: We don't have a practical way**
  - Isolation between applications
  - Support for legacy applications and OS's stack
  - High performance

MultiStack: operating system support for user-space stacks



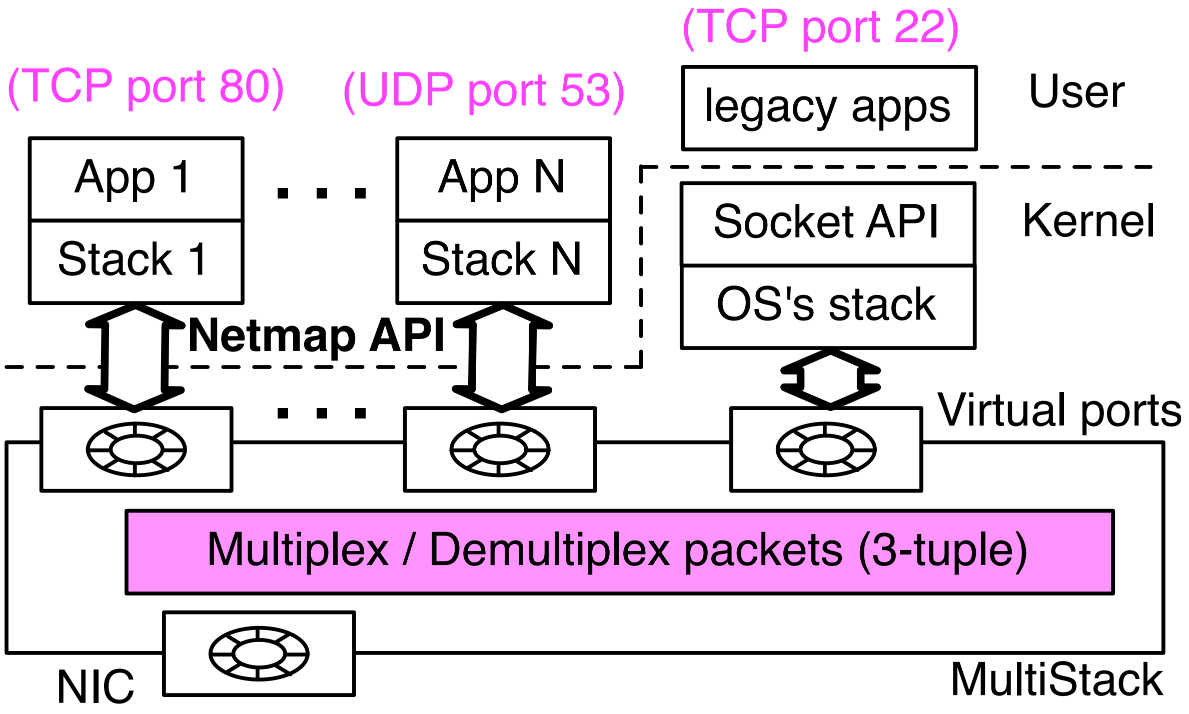
- Support for multiple stacks (including OS's stack)
- Namespace isolation based on traditional 3-tuple
- Very high performance
- Run in FreeBSD and Linux

MultiStack: operating system support for user-space stacks



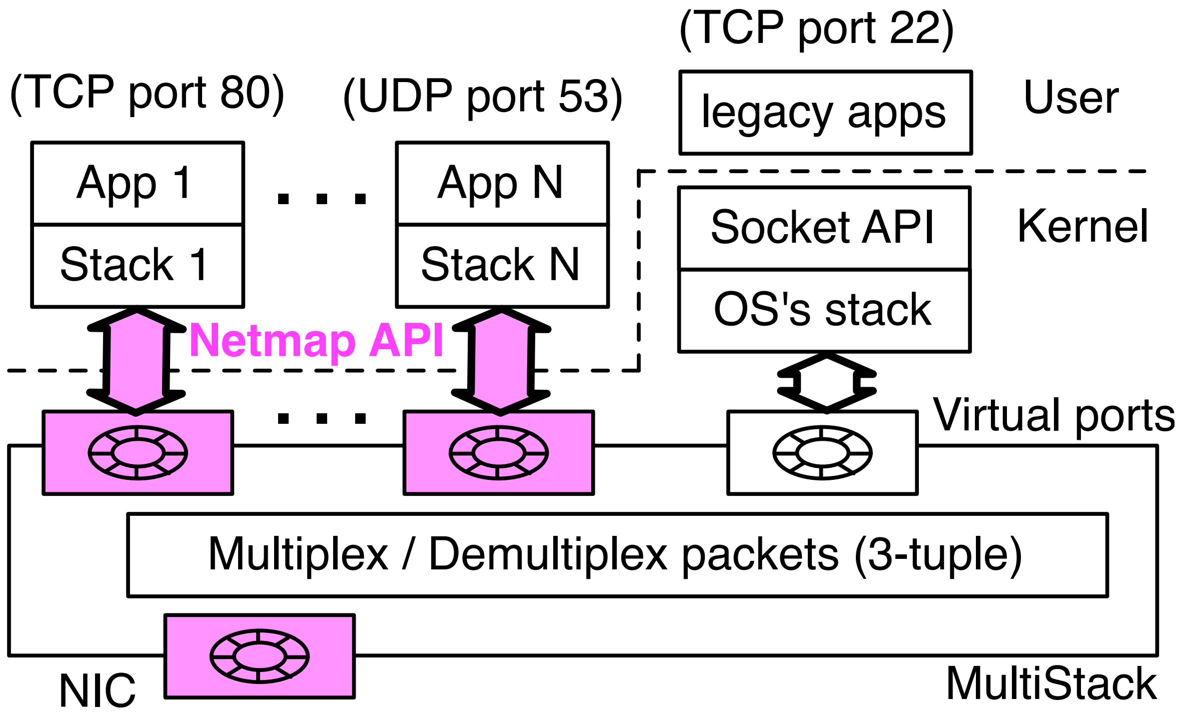
- Support for multiple stacks (including OS's stack)
- Namespace isolation based on traditional 3-tuple
- Very high performance
- Run in FreeBSD and Linux

MultiStack: operating system support for user-space stacks



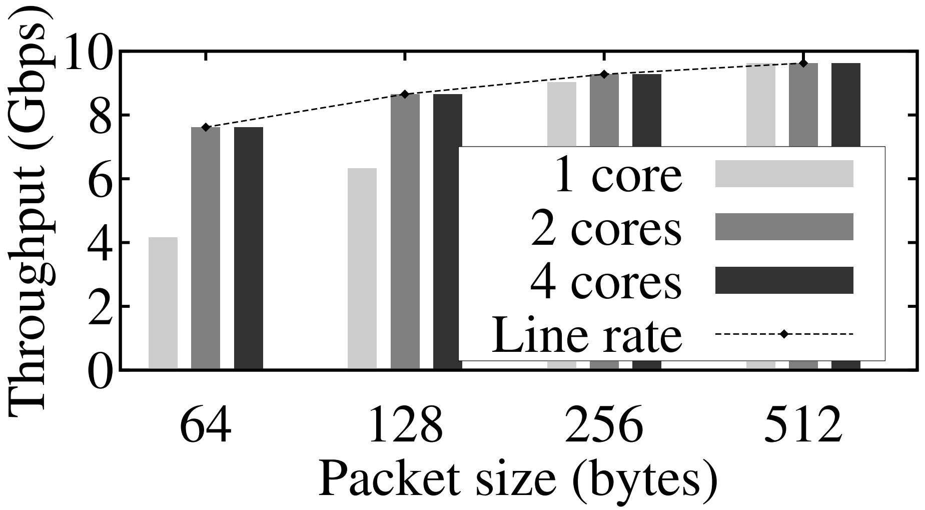
- Support for multiple stacks (including OS's stack)
- Namespace isolation based on traditional 3-tuple
- Very high performance
- Run in FreeBSD and Linux

MultiStack: operating system support for user-space stacks

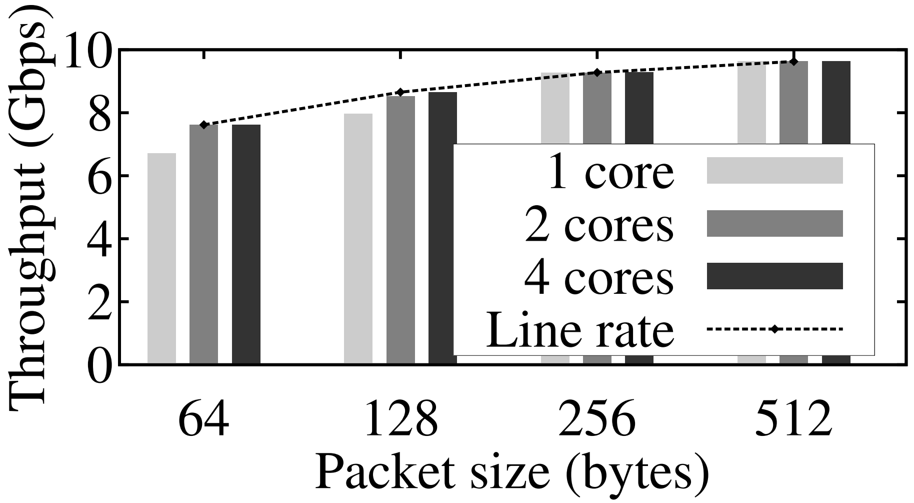


- Support for multiple stacks (including OS's stack)
- Namespace isolation based on traditional 3-tuple
- Very high performance
- Run in FreeBSD and Linux

Multistack performance



TX



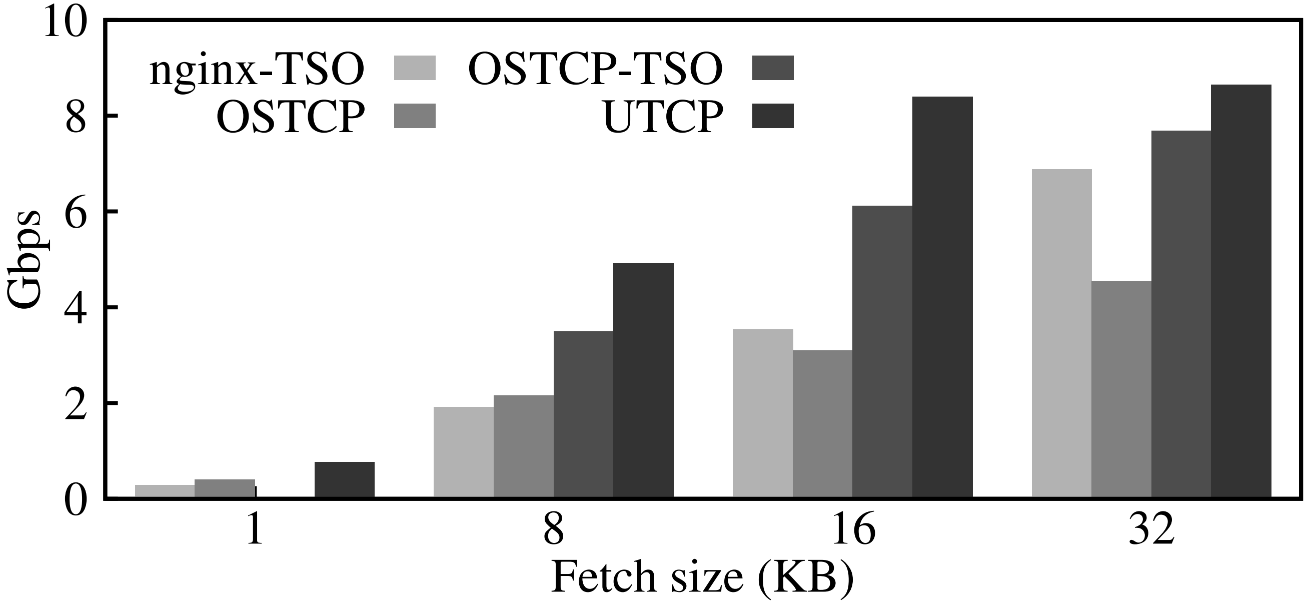
RX

- App creates every packet from scratch, and send it to the kernel
- Multistack validates the source 3-tuple of every packet, and copies the packet to the NIC's TX buffer

- Multistack receives a packet
- It identifies destination 3-tuple of the packet
- It delivers the packet to the corresponding app/stack

Performance with stacks

- A simple HTTP server on top of our work-in-progress user-space TCP (UTCP)
- The same app running on top of OS's TCP



Client establishes a TCP connection, and sends HTTP GET

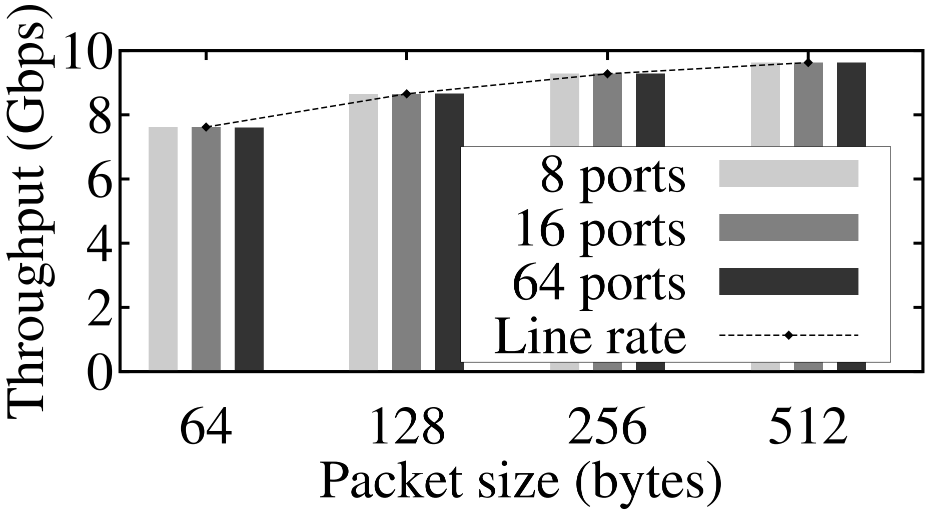
Server replies with HTTP OK (1- 32KB)

Single TCP connection is used for a single HTTP transaction

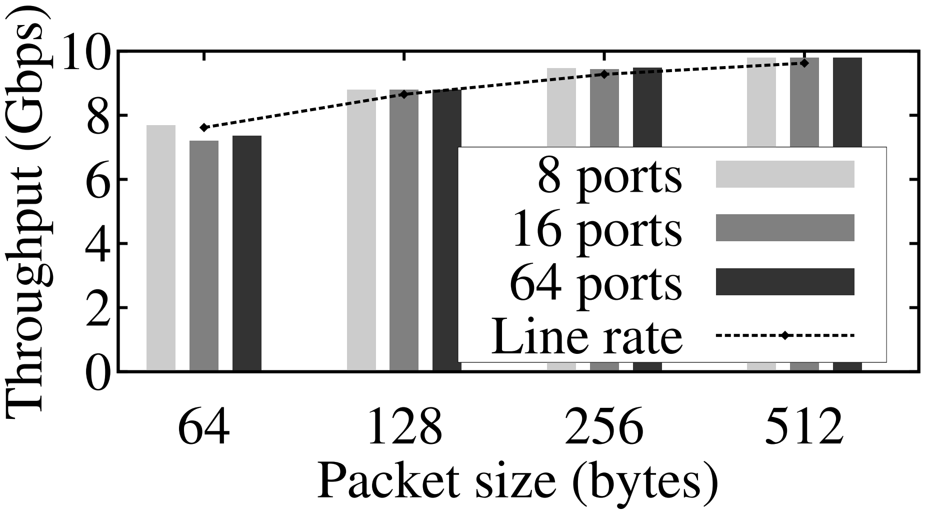
Conclusion

- Rekindle user-space stacks for widespread, timely deployment of new protocols/extensions
- Ongoing work:
  - Improving implementation of MultiStack
  - Making complete user-space TCP implementation
  - Integrating user-space stacks into networking library like libevent and libuv

Multistack performance (many apps/ stacks)



TX



RX

- A bit lower performance on many ports is due to the reduced number of packets taken in a single systemcall