# UOA: User-oriented Addressing for Slice Computing

Maoke Chen, Akihiro Nakao, Olivier Bonaventure and Taoyu Li

ITC SS on Network Virtualization, Hoi An, VN
May 18 2009

# Outline

- Background: slice and isolation
- Motivation: isolation in native OS
- UOA for testbed
- UOA for NIRA
- Evaluation: challenges in performance
- Future work

# Slicing = isolation

- The importance of isolation
  - prevent interferences among slivers
  - make every sliver as a logically complete computer
- What is to be isolated?
  - Namespace identifiers for networking in different layers, root environment
  - Security data/file accessibility, behavior visibility, etc.
  - Performance CPU and memory usage, disk quota, bandwidth scheduling

# Technology of Isolation

- Virtualization
  - Hypervisor-based (full virtualization)
    - VMWare: hardware emulation
    - Xen: para-virtualization, modifying guest OS to apply hypercalls through Application Binary Interface (ABI)
    - Advantages
      - Easy to deploy, support for heterogeneous guest OS
    - Disadvantages
      - Heavy overhead, poor performance scalability

# Technology of Isolation

- Virtualization: instrument of isolation
  - Container-based
    - PlanetLab — OpenVZ — EmuLab — NetNS
    - Advantage
      - Lightweight: no overhead of emulation
    - Drawback
      - over-engineered! => limitations
        - conflict with other features (e.g. NetNS vs. sysfs)
        - difficult code maintenance

# A Story

- 2004: CNGI R&D Project was launched
  - Experiment platform was required
    - native IPv6 support
    - native IPv4 and IPv6 multicast support
  - Large scale
    - each project has 3 ~ 15 subtopics of experiment
    - each experiment should be deployed over more than 25 nodes over the country

# A Story

- PlanetLab (MyPLC) didn't support IPv6 nor multicast
  - until late 2008
  - due to the engineering in container
- Similar but a different thing is needed
  - which?

# Isolation Revisited...

- Native OS does also support isolation to some extent
  - security
    - user-oriented file/directory permissions
  - performance
    - process-oriented scheduling in CPU time and memory
    - user-oriented disk quota
    - attribute-oriented scheduling in traffic reshaping ("tc")

# 6PlanetLab

- Slicing based on native OS
  - user identifier (uid) => sliver
  - optional Xen, to support alien guest OS
  - Advantage
    - having all the features supported by native OS
    - easy to deploy, without extra installation and configuration
  - Still missing:
    - network namespace isolation!

# User-oriented Addressing

- Idea
  - making sliver from uid
  - taking native OS exiting facility for security and performance isolation
  - adding network namespace isolation for uid
    - => change in IP addressing model

# IP Addressing Model

- Addressing model
  - defines how Internet addresses are assigned to entities
- Current IPv4 and IPv6 addressing model
  - IP addresses are assigned to hosts or, more exactly, to interfaces

# Architecture of UOA

- Case 1: UOA for Testbed
- Assumptions
  - Existence of a centralized SliceMan
  - Enough IP address space
  - Multi-user OS
  - Root context communicating with SliceMan

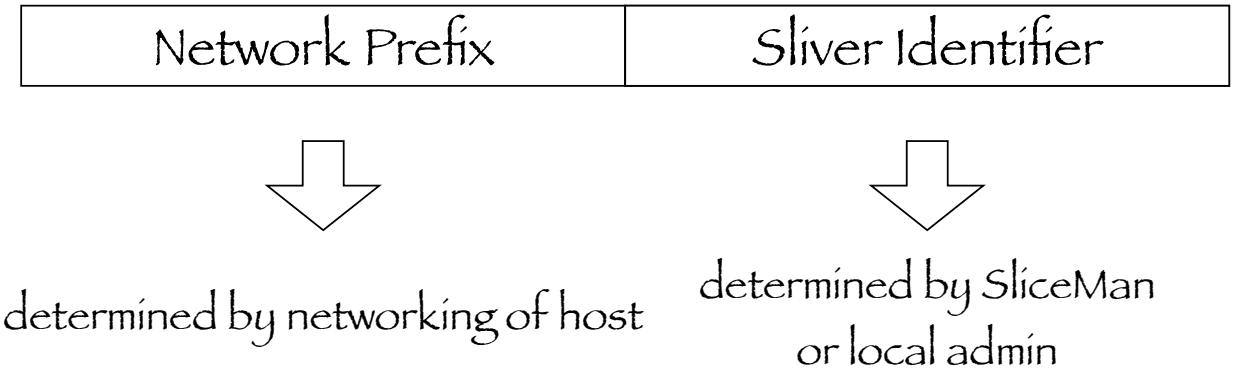
# Architecture of UOA

- Requirements
  - Uniqueness
    - spatial MAC-related mechanism (DHCP, auto-conf) fails
    - Temporal sliver moving among hardware without changing addresses
      - cannot be fully satisfied without EId/Rloc separation
  - Behavior traceability vs. privacy
  - Binary compatibility

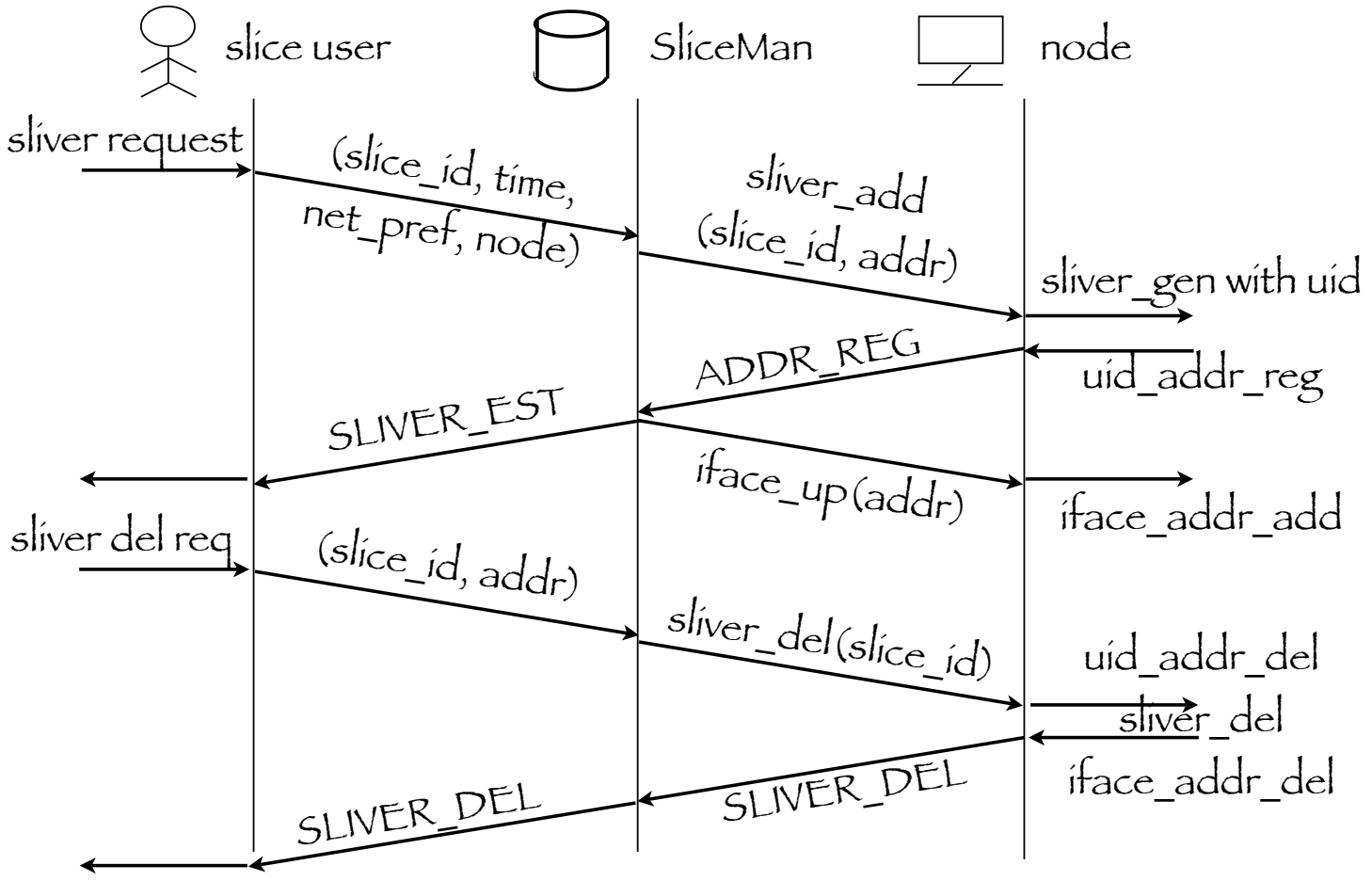
# Address generation

Manageability + Privacy + Uniqueness

Slice Id => uid  hash  time_to_assign

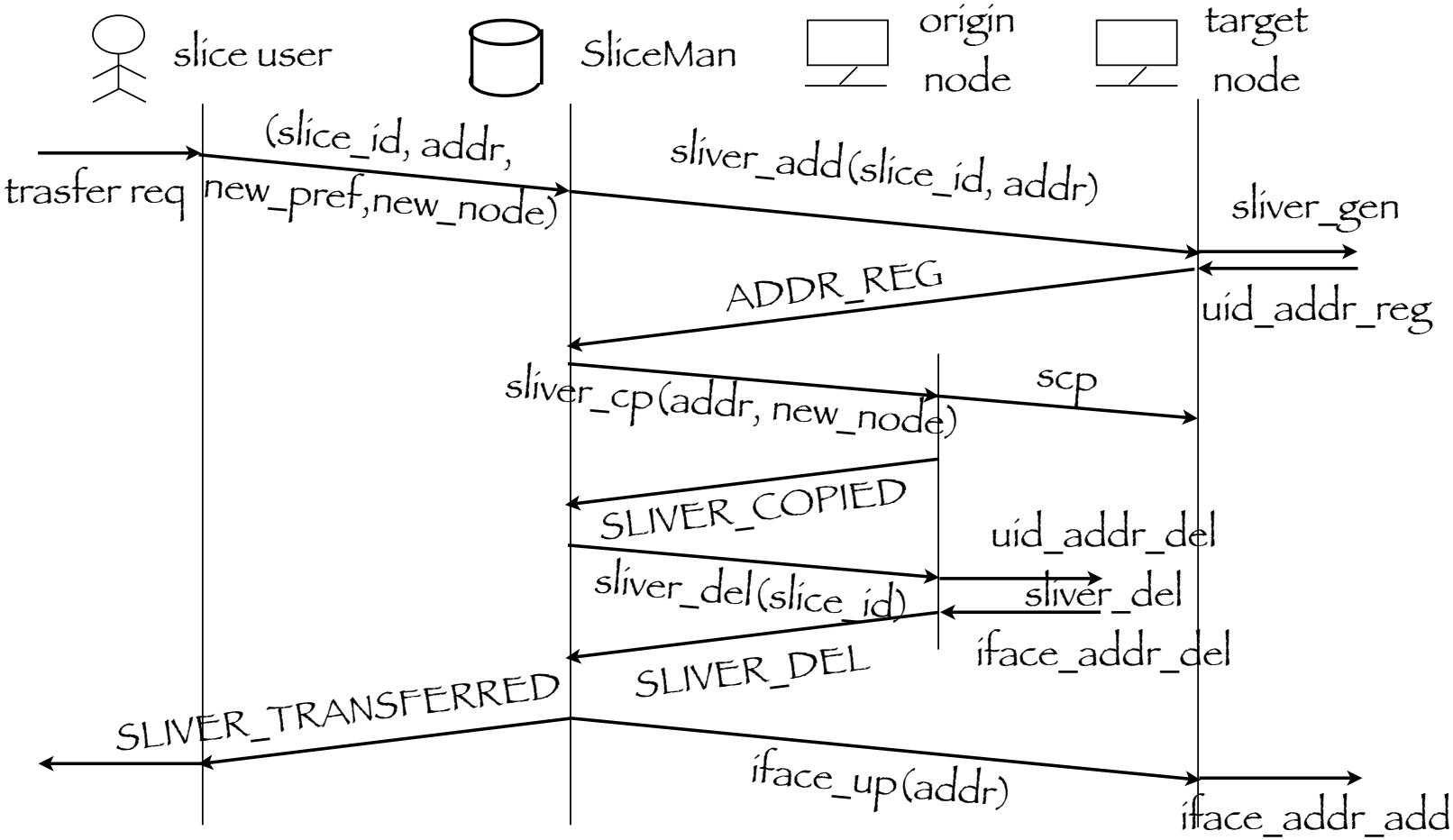


# Address Assignment



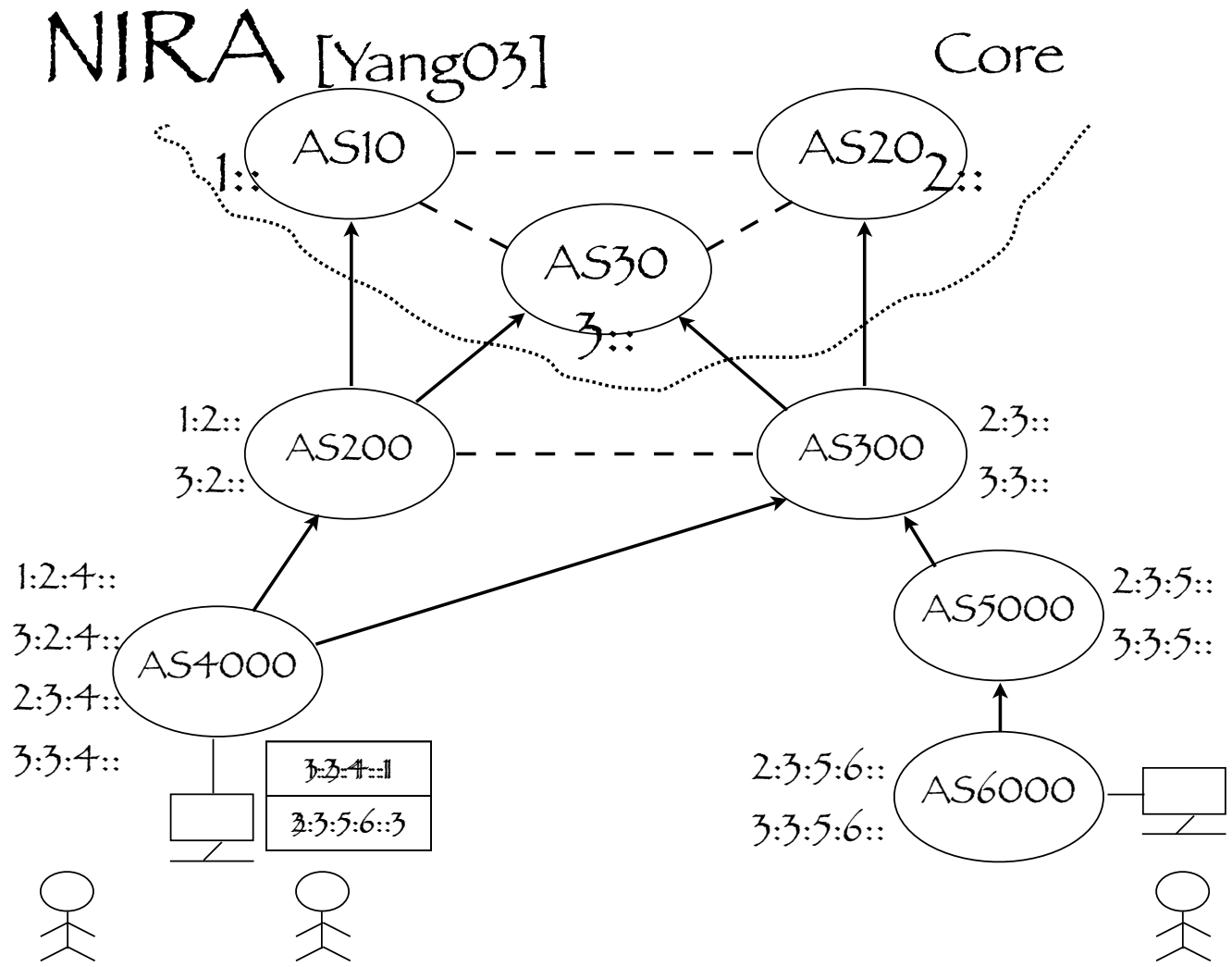
# Sliver Transfer

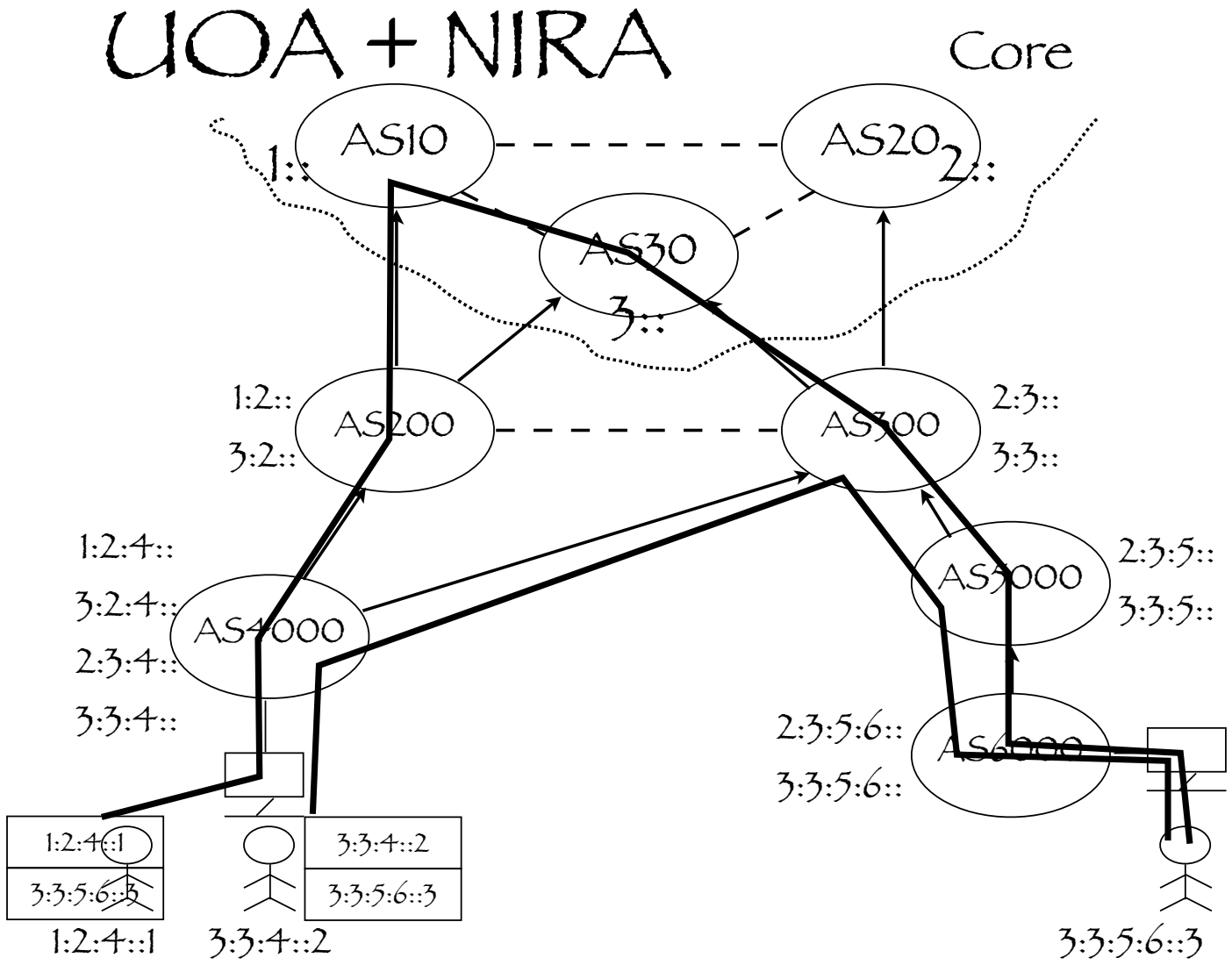
- without changing address
- keeping temporal uniqueness



# Architecture of UOA

- Case 2: fine-grained routing
- New trends in routing architecture innovations
  - involving user control => neutrality (NIRA )
  - split the role of EId/RLoc (LISP, shim6, ...)
- Common idea
  - leveraging routing by changing address
  - ...but host-level leveraging is not enough


NIRA [Yang03]
Core
AS10
1::
AS20
2::
AS30
3::
1:2::
3:2::
AS200
AS300
2:3::
3:3::
1:2:4::
3:2:4::
2:3:4::
3:3:4::
AS4000
AS5000
2:3:5::
3:3:5::
2:3:5:6::
3:3:5:6::
AS6000
3:3:4::1
3:3:5:6::3


UOA + NIRA
Core
AS10
AS20
1::
2::
AS30
3::
1:2::
3:2::
AS200
AS300
2:3::
3:3::
1:2:4::
3:2:4::
2:3:4::
3:3:4::
AS4000
AS5000
2:3:5::
3:3:5::
2:3:5:6::
3:3:5:6::
AS6000
1:2:4::1
3:3:5:6::5
3:3:4::2
3:3:5:6::3
1:2:4::1
3:3:4::2
3:3:5:6::3

# Easy to be Implemented?

- Changes
  - Source address check for connect and bind

    uid -> IP address
  - Semantics of wild-card address
    - impacts both port selection and packet dispatching

      socket -> uid

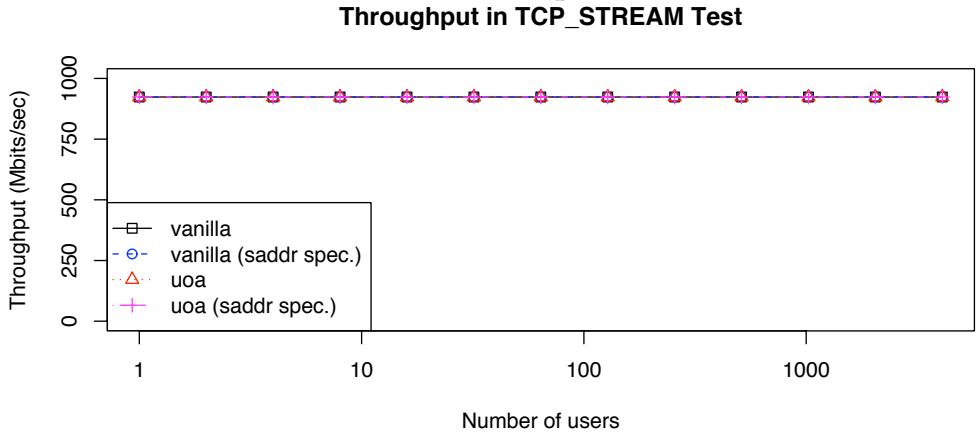
# UOA Implementation

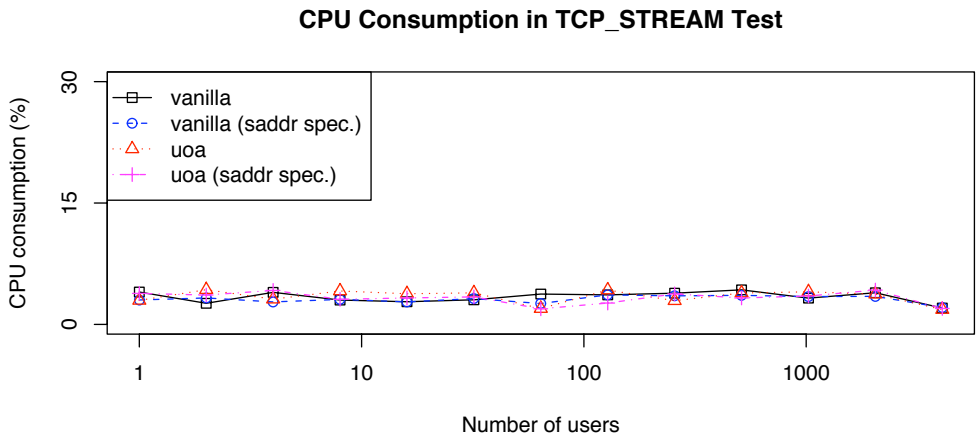
- Kernel patches
  - Linux 2.6.x
  - FreeBSD 5.0
- Interfaces
  - administration tool
  - procfs view

# Performance Concern

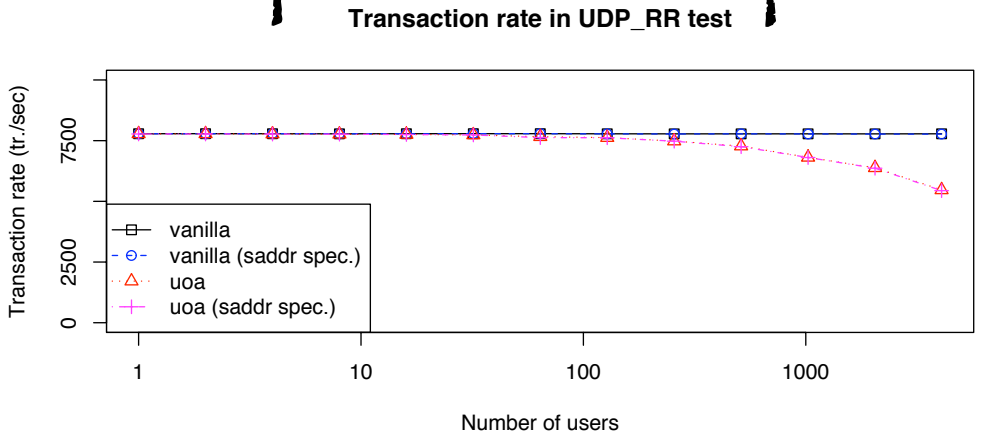
- Benchmarking tool
  - netperf
- Goal
  - performance impact of UOA codes in comparison to vanilla OS

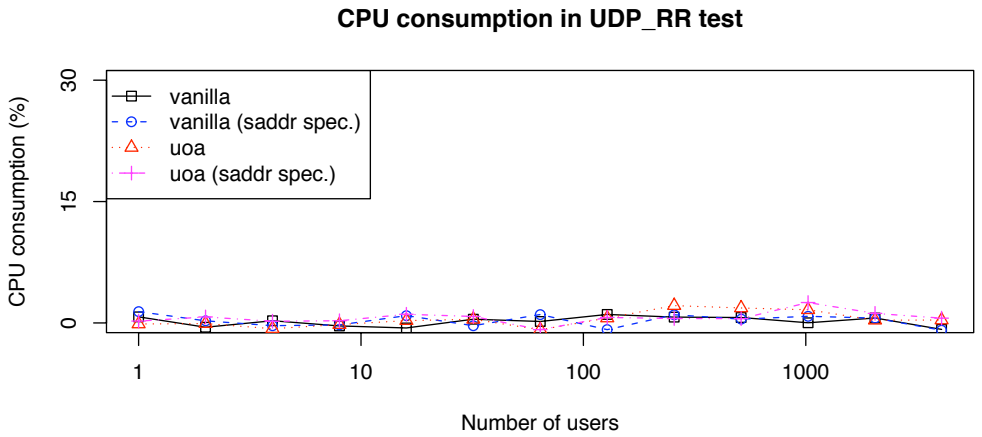
# TCP Streaming



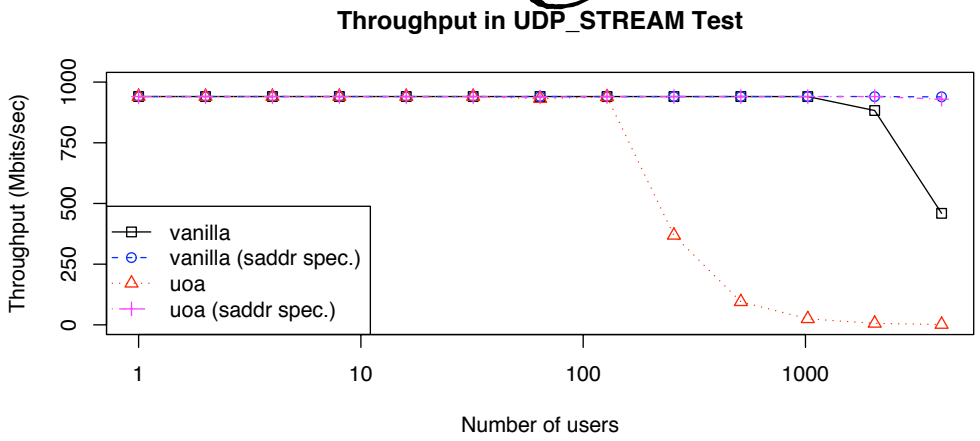



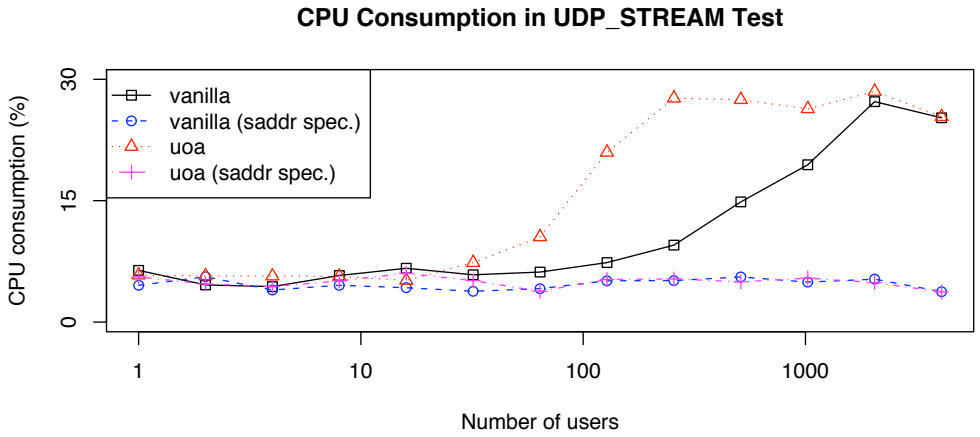
# UDP Request/Response



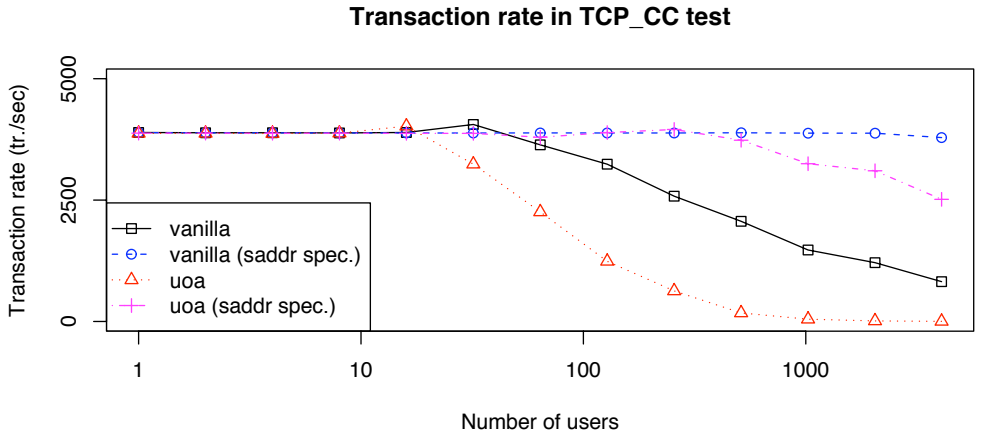



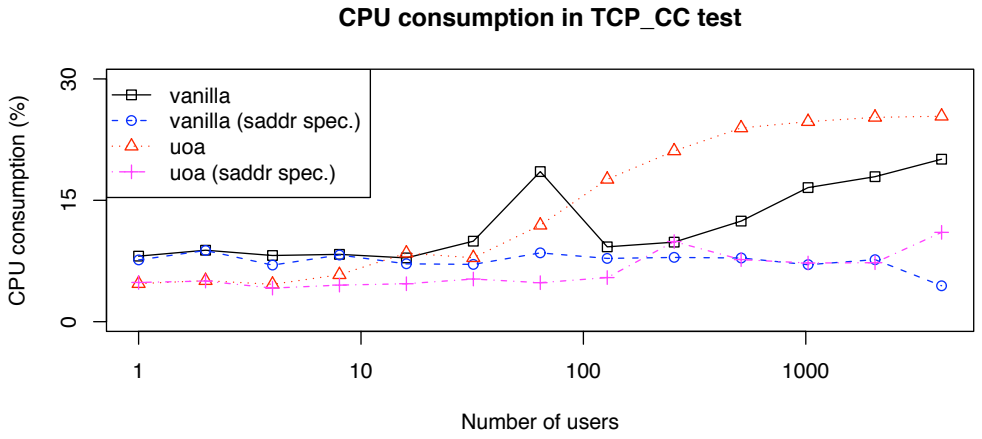
# UDP Streaming



# TCP Connection/Close



# Interpretation of Bench.

- Wild-card address involves more overheads (UDP RR) <= port selection goes through all sockets in binding
- Repeatedly checking the uid->address mapping is CPU consuming (UDP STREAM)

$$O(|U|C(|U|))$$

- Unclosed sockets make things heavier (TCP CC, due to TCP TIME_WAIT sockets)

$$O(|V|(1+\rho(|V|))(C(|U|)+C(|V|))$$

# Improvement

- Using hash tables instead of linked list
  => reduce the $C(\cdot)$
- Coupling socket hash buckets with user-oriented assignment information
  => avoid repeated retrievals

# Conclusions

- Philosophy: isolation with commodity OS plus minimum add-on
  - making things as built-in as possible
    - easy deployment and configuration
    - no conflict with other features
    - code simplicity for easy maintenance
  - encourage popular users to join slice

# Future Work

- Can uid support isolation as much as possible?
  - network namespace: UOA + PBR => routing isolation
  - performance: UOA + tc => bandwidth isolation
  - performance: cgroup => CPU time isolation/scheduling
  - ……