IX: A Protected Dataplane Operating System
Problem Context

- The requirements of modern data center applications are being redefined:
  - High packet rates
  - Quick responses to remote requests
  - Support for high connection counts
  - Strong protection model
  - Elastic resource usage
Problem Context: (continued)

- Typically, aggressive network requirements are handled in the user-level networking stack...
  - This mitigates the mismatch between requirements and the network stacks in commodity OSes
  - Reduces some overheads, doesn’t handle other trade offs
  - Networking stack not protected!
The Objective:

To improve operating systems for applications with aggressive networking requirements that run on multi-core servers.
Problem Context: Importance

Search engines, social networks, and e-commerce platforms all have the aforementioned aggressive network requirements.

Improving OSes to handle these types of requirements will have far-reaching benefits, and will have a positive impact on these applications.
Related Work: Hardware Virtualization

**Arrakis:** An OS that uses hardware virtualization to separate the I/O data plane from the control plane.

- Applications have direct access to virtualized I/O devices
- Kernel is re-engineered to provide network and disk protection without constant mediation
Related Work: Library Operating Systems

**Exokernel**: reduces the functionality in the OS kernel to bare minimum, and punts most of the logic to user space. Provides basic access to resources like memory or disk, but does not include resource management.

**IX**: limits itself to the implementation of the networking stack, allows applications to implement their own resource management policies.
Related Work: Async/Zero-Copy Communication

Systems with async, batched, or exception-less syscalls reduce context switch overheads. Zero-copy reduces data movement overheads, and simplifies resource management.

**POSIX OSes:** support zero-copy through page remapping and copy-on-write.

**TinyOS:** passes pointers to packet buffers between the network stack and the application in a zero-copy way.
IX: The Solution

Breaks the 4 way tradeoff between high throughput, low latency, strong protection, and resource efficiency.

Separates control plane and dataplanes:

- **Control plane**: system configuration, coarse-grain resource provisioning
- **Data planes**: run the networking stack and the application logic
IX Design: Control Plane

- Full Linux kernel:
  - Initializes NICS
  - Provides mechanisms for resource allocation to dataplanes
  - Provides syscalls and services

- IXCP (a user-level program):
  - Monitors resource usage and data plane performance, and implements resource allocation policies
IX Design: Data Plane

- Each one supports a single, multithreaded application
- Each operates as a single address-space OS, and supports two thread types
  - Elastic threads (interact with the IX dataplane to initiate and consume network I/O)
  - Background threads
  - Adaptively set batch size
IX Dataplane: Batching

- Dataplanes run all stages needed to receive and transmit a packet to completion, regardless of load conditions
  - No need for intermediate buffering between protocol stages and between app logic/networking stack
- Improves message throughput and latency because successive stages tend to access the same data as those prior to them
  - Improves data cache locality
- Adaptive batching:
  - Never wait to batch requests, batching only occurs in the presence of congestion
  - Set an upper bound on the number of batched packets
IX Design: Dataplane

- Designed around a native, zero-copy API
  - Supports processing bounded batches of packets to completion
- Network adapters (NICs) redirect incoming traffic to distinct queues through hashing
- Each dataplane instance exclusively controls a set of these queues
- Runs the networking stack and application
IX: Security Model

- IX protection model doesn’t assume anything about the behavior of the application
- A malicious/misbehaving application can only hurt itself
  - Can’t corrupt the networking stack
  - Can’t affect other applications
- All application code runs in user mode
- Dataplane code is protected
- Applications can’t access dataplane memory, except for read-only message buffers
What Dataplanes Do Well

- API overheads are amortized
- Data and instruction locality is improved
- Increased throughput, decrease latency
- Optimizes for multi-core scalability
Results

● Comparing IX with a TCP/IP dataplane against Linux and mTCP
  ○ On a 10 GbE experiment: IX outperforms both by up to 10x and 1.9x respectively for throughput

● Unloaded uni-directional latency for two IX servers is 4x better than between standard Linux kernels and an order of magnitude better than mTCP
Results: (continued)

(a) Multi-core scalability (n=1, s=64B)

(b) n round-trips per connection. (s=64B)
Limitations

- Fragmentation and inefficient resource utilization is one of the primary drawbacks of this system.
- Coarse-grained allocation of large chunks of resources reduces allocation overhead.
  - Tradeoff is that resources are underutilized because more resources are required for otherwise the same workload.
Next Steps in the Space

Need to develop efficient resource allocation policies, which requires the understanding the tradeoffs between:

- Data plane performance
- Energy proportionality
- Resource sharing between co-located applications with varied workloads