Protocol Implementation

An Engineering Approach to Computer Networking
Protocol implementation

- Depends on *structure* and *environment*

- Structure
  - *partitioning* of functionality between user and kernel
  - separation of layer processing (*interface*)

- Environment
  - data copy cost
  - interrupt overhead
  - context switch time
  - latency in accessing memory
  - cache effects
Partitioning strategies

- How much to put in user space, and how much in kernel space?
  - tradeoff between
    - software engineering
    - customizability
    - security
    - performance
- Monolithic in kernel space
- Monolithic in user space
- Per-process in user space
Interface strategies

- Single-context
- Tasks
- Upcalls
Monolithic in kernel
Monolithic in user space
Per-process in user space
Interfaces

- Single-context
- Tasks
- Upcalls
Single context
Tasks
Upcalls

- SEP = SEND ENTRY PT.
- REP = RECEIVE ENTRY PT.
- RGP = REGISTRATION ENTRY PT.

Diagram showing the flow of data through different layers:
- Application
- Session
- Transport
- Network
- Datalink/Device Driver
Protocol implementation
Some numbers

- 10 Kbps       400 ms
- 100 Kbps,     40 ms
- 1 Mbps,       4 ms
- 100 Mbps,     40 µs
- User-to-kernel context switch ~40 µs
- Copying the packet ~25 µs
- Checksum in software ~40 µs
- Scheduling delays ~150 µs (depends on workload)
- Interrupt handling ~10-50 µs (depends on the bus)
- Protocol processing ~15 -100 µs (depends on protocol complexity)
Rules of thumb

- Optimize common case
- Watch out for bottlenecks
- Fine tune inner loops
- Choose good data structures
- Beware of data touching
- Minimize # packets sent
- Send largest packets possible
- Cache hints
- Use hardware
- Exploit application properties