High Performance Networking

-- Increasing performance in GB applications

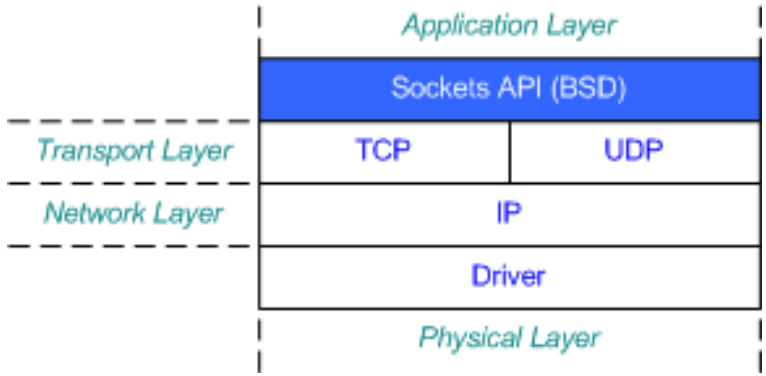
M. Tim Jones
Emulex Corp.

Agenda

- Networking Architecture
- Sockets Programming Introduction
  - API, Clients, Servers, Symmetry
- Where's the Problem?
- Enhancing Performance
  - Application Layer Enhancements
  - TCP/IP Stack Selection
  - TCP/IP Stack Configuration
  - TCP/IP Stack Modifications
  - Specialized Hardware
  - Other Methods
- ~Q&A

Networking Architecture

- Application Layer
  - Networking App
- Sockets Layer
  - Developer's Interface
- Transport Layer
  - TCP, UDP, SCTP, …
- Network Layer
  - IPv4/IPv6, IPSec

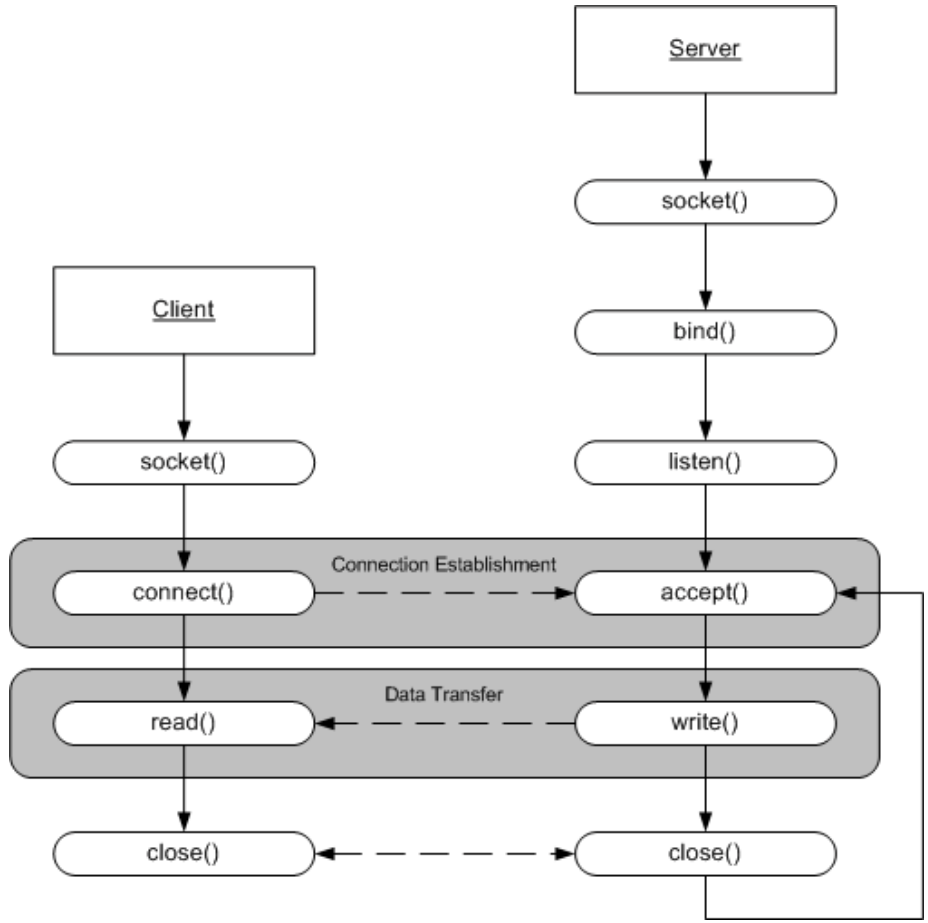


Sockets Programming

- Simple API provides primitives for client and server implementation
- Provides multi-protocol support (TCP/UDP/SCTP/…)
- Provides application-layer configuration (behavioral, performance, etc.)

TCP Sockets Symmetry

- Relationship between calls in application
- Relationship between client and server
- Performance aspects



Enhancing Performance

Where’s the Problem?

Where's the problem?

- Data Movement
  - Buffer Copies
- Data Manipulation (Touching the data)
  - Validation, Checksums, Encryption, Decryption.
- Resource Management
  - Locks, Cache, Connection contexts
- Protocol Overhead
  - Fragmentation, Reassembly, Flow Control, Congestion Control
  - 1Hz for 1bps (Intel Rule)

Enhancing Performance

Application Layer Enhancements

Application Layer Enhancements

- Nagle
- Bandwidth Delay Product
- Read/Write calls
- Control and Data Connections
- Select API Function
- Network Striping (Multiple Connections)
- TCP vs. UDP

Nagle

- Minimize small packets (tinygrams) on the network by coalescing data
  - Enabled by default
- Disabling permits small packets to be generated
  - Decreases latency (<200ms)
- Enabled via the TCP_NODELAY socket option

```
int option = 0;

ret = setsockopt( sock, IPPROTO_TCP, TCP_NODELAY,
                  (void *)&option, sizeof(option));
```

BDP

- Bandwidth Delay Product
  - Product of round-trip time and estimated minimum bandwidth between endpoints
- TCP Buffer sizes set accordingly
  - Buffer size determines maximum amount of unacknowledged data that can be sent (simplified)
- Example:
  - 1ms RTT * 1Gb/s link speed = 125KB

BDP (cont)

- Adjusted through setsockopt:

```
int bufsize= 65536;

ret = setsockopt( sock, SOL_SOCKET, SO_RCVBUF,
                  (void *)&bufsize, sizeof(bufsize));

ret = setsockopt( sock, SOL_SOCKET, SO_SNDBUF,
                  (void *)&bufsize, sizeof(bufsize));
```

- Must be done before connection
  - Client before connect
  - Server before accept (child socket inherits)

BDP (cont)

- 64K windows not large enough for LFN or High-Speed Networks.
- To support RFC1323, window scale option provided (shift base socket buffer size)
  - Supports up to 1GB window

Read/Write calls

- Write as much data / Read as much data as possible per call
  - Reduction in kernel context switches
  - Minimize the number of buffer copies (write)
  - Keeps advertised window open (read)
- If data is packet oriented (header + data), peeking the header and then reading the total size can sometimes be beneficial
  - Can help in message framing

Control/Data

- Provide separate connections for control and data
  - Control connection used solely for command/response
  - Data connection used for bulk data transfer
  - Characteristics of connections different
- Successful with File Transfer Protocol (FTP)
  - Telnet-like connection for control
  - Data connection for bulk data

Avoiding select

- select commonly induces poor performance
  - Kernel mods
  - Sam?
- Alternatives?
  - Event Callbacks (if available)
  - Stack modifications

Network Striping

- Open multiple connections and stripe data over them
  - Improves bandwidth utilization
  - Reduces latency
  - Operates on un-optimized sockets (small windows, etc.)
- PSockets at University of Illinois

TCP vs. UDP

- To avoid computational complexity of TCP, use raw UDP?

- Rarely beneficial (from my experience)
  - TCP includes complexity for a reason
  - Can work for very simple protocols (idempotent packet transfer)
    - Common broadcast / multicast

Enhancing Performance

TCP/IP
Stack Selection

TCP/IP Stack Selection

- RFC Compliance
  - Standards…
  - RFC1323
  - RFC2001
- Zero-Copy APIs
- Event Registration / Socket Callbacks
- Timer Implementation
- Resource Management
- Reconfigurability
- Commented-Source
- ANVL

Enhancing Performance

TCP/IP
Stack Configuration

TCP/IP Stack Configuration

- Compile-time reconfiguration
  - Disable unnecessary functionality
- Tune resource availability
- Avoid memory heaps
  - Resource queues as an alternative
- Scenario-Specific Configuration
  - Direct-routes
    - No PMTU Discovery
    - No Routing Decisions

Jumbo Frames

- Goal: Largest link MTU not exceeding Path MTU
- Increased MTU means more payload per segment
  - Typical 1500 MTU typically means 1448 MSS (payload)
  - Typical jumbo frame is 9000 MTU
- Path MTU Discovery (RFC 1191)
  - Used only for indirect routes
- MSS Negotiation (during three-way handshake)

Jumbo Frames (cont)

- What do jumbo frames mean to the stack?

Rate
10 Mb/S
100Mb/S
1Gb/S
10Gb/S

MTU	Packet Time	Packets/ Second
1500	1200uS	833
1500	120uS	8333
1500	12uS	83333
1500	1.2uS	833333

MTU	Packet Time	Packets/ Second
9000	7246uS	138
9000	720uS	1388
9000	72uS	13888
9000	7.2uS	138888

- Less processing per packet

Enhancing Performance

TCP/IP
Stack Modifications

TCP/IP Stack Modifications

- Two types of modifications
  - TCP Friendly
    - Changes that interoperate with unmodified stacks
  - Non-friendly
    - Changes that require symmetric changes (ECN, etc.)
- We'll focus here on some TCP-friendly changes

TCP/IP Stack Modifications

- Delayed Ack Removal
- Minimizing Acks
- Slow-Start
- Scalable TCP (RFC-2581)

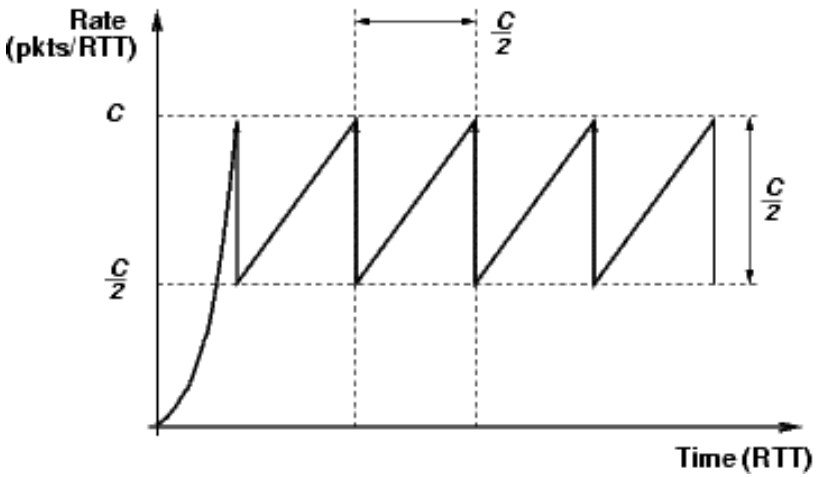
Scalable TCP

- Flow Control
  - Receiver via advertised window
  - Sender via congestion window
- Scalable TCP alters functions that manipulate the congestion window (for large windows)

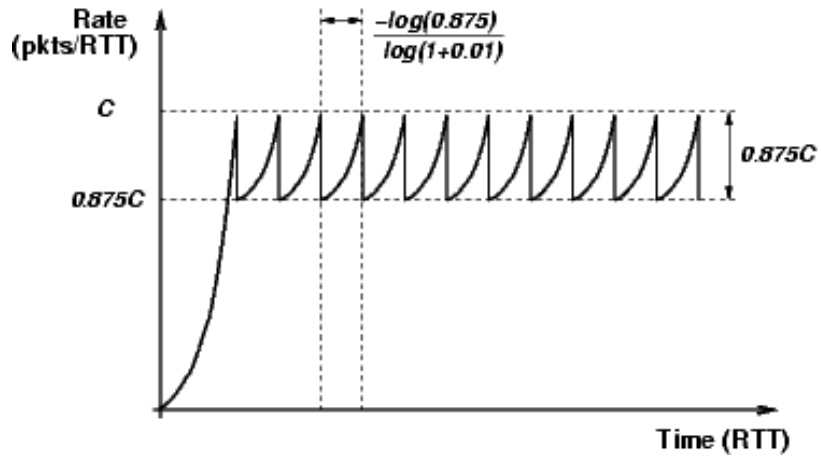
	Typical Algorithm	Scalable TCP
ACK Received	$cwnd = cwnd + \frac{1}{cwnd}$	$cwnd = cwnd + 0.01$
Congestion Detection	$cwnd = \frac{cwnd}{2}$	$cwnd = cwnd - (0.125 * cwnd)$

Scalable TCP (cont)

- Recovery at 10Gbps
  - ~5 hours



- Recovery at 10Gbps
  - 2.7 seconds



- http://www-lce.eng.cam.ac.uk/~ctk21/scalable/

Enhancing Performance

Specialized Hardware

Specialized Hardware

- TCP Offload Engines (TOE)
  - Partial Offload NIC
  - Full Offload NIC
  - Protocol Acceleration
- TOE Chips
- Encryption/Decryption Chips
  - IPSec

Enhancing Performance

Other Methods

Other Methods

- Stream Control Transmission Protocol (SCTP)
  - RFC-2960
  - Associations
  - Multi-homing
  - Framing
  - Unordered Delivery
- Real-Time Transport Protocol (RTP)
  - RFC-1889
  - UDP-based
  - RTP Control Protocol (RTCP)
  - QoS handled separately (using protocols such as RSVP)

Other Methods

- Remote DMA Protocol (RDMA) + DDP
  - Eliminates copies between kernel and application
  - Explicit buffer definition
  - Good use of resources
  - Can be used by TCP, SCTP, others
  - Not yet a ratified IETF spec

Future

- Socket Instrumentation (network-aware applications)
  - getsockopt()
    - RTT
    - Utilized Bandwidth
  - Socket Buffer Tuning
- Automatic configuration (network-aware operating systems)
  - Tune individual socket performance
  - Tune stack performance
    - Number of connections, available resources, available BW
- Better tools (monitoring and analysis)

Enhancing Performance

- Final Questions