WHITE PAPER

October 2013

# Accelio – The OpenSource I/O, Message, and RPC Acceleration Library


Overview ..... 1
Key Features and Capabilities ..... 1
Architecture ..... 2
Application Examples ..... 5
Comparing Accelio to Traditional Socket APIs ..... 7
Summary ..... 8


## Overview

The increased industry focus on scale-out compute, storage, and application architectures requires an easy-to-use common messaging and data transfer layer. This layer is required to deliver maximum performance, lowest-latency, and to support modern server architectures and large multi-core systems. In addition, the layer needs to fulfill the requirement for end-to-end reliable transaction delivery.

Accelio is an Open Source high-performance, asynchronous, reliable messaging and Remote Procedure Call (RPC) library. Accelio optimizes hardware acceleration and/or Remote Direct Memory Access (RDMA). The library supports other transport implementations such as TCP/IP and shared-memory. Accelio maximizes message and CPU parallelism, while minimizing CPU contention and locking. The parallel and asynchronous architecture, lockless design, and zero data copy mechanism provide unparalleled transaction per second and bandwidth performance, as well as lower latency and CPU overhead. Accelio guarantees end-to-end transaction delivery and execution, and it supports a transactional request-replay communication model.

Accelio addresses challenges of scale-out and virtualized environments by providing built-in multi-pathing, live session redirection/migration, and service or storage clustering. Managed as a modular Open Source project, Accelio extends seamlessly to new functionalities, transport implementations, and services.

## Key Features and Capabilities

- Simple and abstract API for high-performance asynchronous communication and data delivery
- Reliable end-to-end message delivery
- Request/Reply or Send/Receive models
- Connection and resource abstraction to maximize scalability and availability
- Zero copy data delivery engine, with optional built-in memory management
- Maximizes the benefits of RDMA, hardware offloads, multi-core CPUs, and multi-threaded applications
- Supports multiple transport options
- Seamless integration with common event loop mechanisms such as epoll, libevent, and ACE
- Fast event notifications, optional busy wait polling, or combined models for lowest message latency
- Active/Active multi-path support for high-availability and network scalability
- Built-in security and AAA (Authentication, Authorization, and Accounting) support

- Native support for service and storage clustering
- Message combining and batch message processing optimization
- User space and kernel implementations, bindings to high level languages

## Architecture

Accelio's primary layers are:

- **Application Interface –** Provides easy-to-use primitives for fast and reliable asynchronous message queue or RPC
- **Connection and Session Management –** Delivers reliable end-to-end connectivity to peer end-points, with dynamic connection establishment, pooling, fault recovery, and migration/redirection
- **Pluggable Transport Layer –** Enables mapping to different hardware or software transport implementations

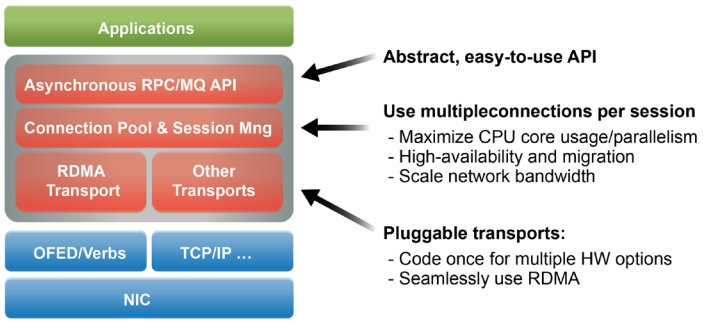


***Figure 1.*** *Accelio Architecture*

### Sessions and Connection Initiation Model

When establishing connectivity to a remote endpoint, clients specify the remote Uniform Resource Identifier (URI). The URI contains the preferred transport, mandatory leading IP address and optional port number, and optional resource ID or name.

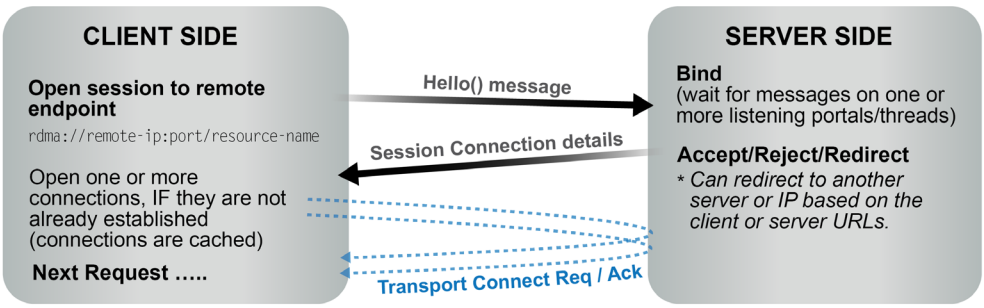


***Figure 2.*** *Accelio Connection Initiation*

The server side responds to a session request with one of the following options:

- Accept the connection with optional properties
- Reject the connection
- Seamlessly redirect the connection to a different process or entity

Following a successful or redirected response, the client establishes one or more parallel connections to the peer, trying to reuse existing connections to minimize overhead.

### Transaction Delivery API Model

Accelio provides an RPC-like request/replay transactional model or a reliable message send/receive model.

In the transactional model, the initiating side issues a request that consists of its private application header and pointers to the outgoing and incoming data buffers to allow zero-copy operation and RDMA.

When the request arrives to the receiving side, it triggers a call-back notification, based on which the receiving application processes the transaction. When the receiving side completes the processing, it sends the response with the returned status and data. The server side can operate asynchronously and issue the response at its convenience, that is, when the returned data is available.

The initiating side may request message arrival acknowledgement. In such a case, an acknowledgement message is returned to the initiator **after** the receiving side accepts the message. The acknowledgement message can be used for barrier/synchronization operations and message tracking.

The initiating side is notified when the response arrives. At this point the initiator data buffers contain the returned data, and the response call-back points to the original request.

If there is a transaction failure, the same transaction is re-transmitted over a recovered or alternate connection.

The initiating side can request to abort/cancel a range of messages (for example, a timed-out request). The receiver side tries to abort the operations (if they were not executed) and respond with a success or failure to abort. Either way, the associated responses are not returned to the initiator.

Note that requests can be issued by either the client (passive) or server (active) side.

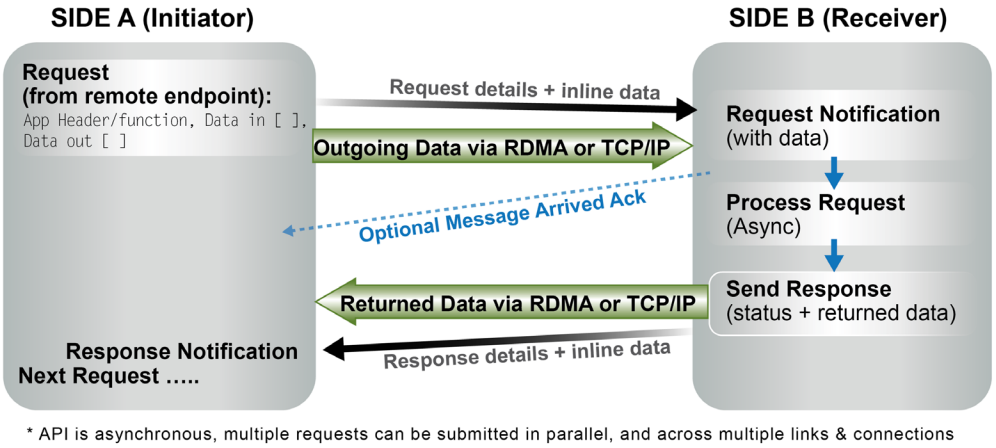


***Figure 3.*** *Accelio Transaction Delivery*

Send/receive operations follow a similar flow to request/response, with the exception that there is no response message. A message is sent from the initiating side, triggering a call-back on the receiving side, with the optional acknowledgement notifications.

### Parallel and Multi-Threaded Execution Model

Accelio supports full parallel and multi-threaded operations for client and server sides

- Each thread can be associated with separate hardware resources (QPs and CQs) and event loops, eliminating the need for locking and allowing maximum parallelism.
- Each session can operate multiple parallel connections, in which each session is associated with a different CPU thread.
- Server side can expose multiple portals (IP:port) for every resource/service, and each portal can be served by a different CPU thread and event loop.

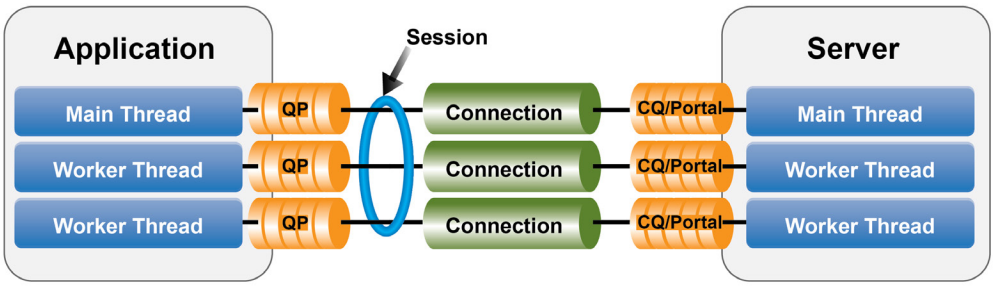


***Figure 4.*** *Accelio Native Multi-thread Support*

### Asynchronous Event Handling

High-performance applications use asynchronous APIs to conduct multiple operations and utilize multiple resources in parallel.

- Accelio handles asynchronous events in the following manner:
  1. Place a callback in an event loop which combines events from multiple sources.
  2. Upon an event, issue a set of operations, such as receive message from network and write to disk.
- Accelio has built-in epoll support. An application-supplied event loop is available as well.
- For lowest messaging latency, Accelio provides optimized event handling that combines busy wait pooling when an event loop is in idle state (for a pre-defined time)..

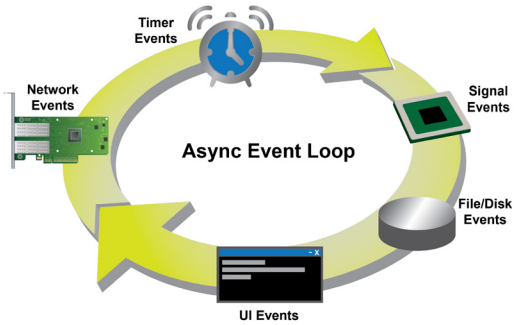


***Figure 5.*** *Accelio Asynchronous Event Handling*

### Memory Management

Accelio manages two types of buffers:

- Send/Receive ring buffers for headers and small data, typically 4-8KB each.
- Large buffers for large IOs and RDMA

The large buffers are allocated and controlled by the library *or* by the application.

- Application-controlled buffers
  - Require pre-registration of the memory buffers (using *reg_mr*), and guarantee zero-copy
- Library-controlled buffers
  - **May** have a copy on the client for sent data
  - Zero-copy on the receiver (app receives a pointer to the received buffer)
  - Require the application to release buffers when finished (using *release_msg*)

### Message Coalescing and Batch Processing

Accelio provides built-in mechanisms to reduce the per message overhead and amortize it across multiple transactions.

- Minimize hardware access and interruptions using smart receive completion management, batch post-receive operations, and send completion coalescing.
- Applications can issue multiple messages that map to single hardware/NIC operations, by using the additional "more" message flags.
- Receiver side provides indications of multiple pending requests/messages, to optimize applications' operations, for example, receive multiple requests and issue a single operation to the disk.
- Aggregation of small messages to a single packet transfer.

## Application Examples

### Remote File and Object Access (R-AIO)

The high speed and parallel nature of Accelio enables reliable high-speed remote file and object storage access with minimal development effort. Accelio provides reliability, high-availability, multi-path, and scale-out clustering tools without requiring complex development.

The R-AIO application demonstrate such a usage model, allowing a standard file access benchmarking tool (Fio) to issue standard file IO requests, which are transferred to and executed on a remote file server, with responses/data returned asynchronously to the client.

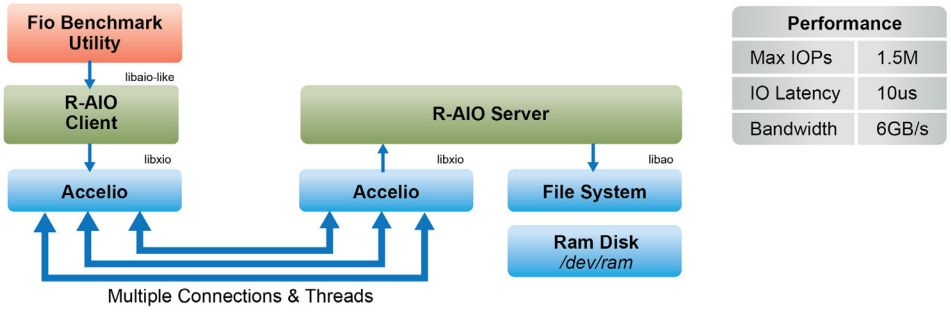


***Figure 6.*** *Using Accelio for Remote File and Object Access*

R-AIO provides a client library that exposes libaio-like APIs that are layered over Accelio, and a server process that intercepts incoming file requests, issues them to its local file system via the libaio API, and returns the response to the client.

For example, tests conducted show that applications can achieve 1.5 million remote file access transactions per second using four CPU threads when accessing a remote RAM disk (/dev/ram), and with a minimum roundtrip latency of under ten microseconds.

### Application and Data Clustering

Accelio delivers an optimal solution for application and data clustering:

- Latency as low as ~4 microseconds for a full transaction (request + response), and consistent low latency even under heavy load
- Low CPU overhead with minimal CPU blocking and maximum parallelism
- High transaction rate and unlimited bandwidth
- Transaction-level reliability
- Network parallelism and multi-path

Accelio can achieve all this with minimal development effort for the application vendor.

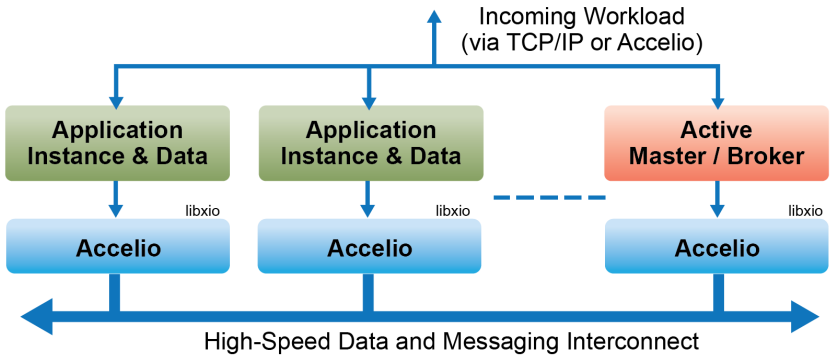


***Figure 7.*** *Using Accelio for Application and Data Clustering*

Accelio has built in facilities for session and connection redirection and dynamic service location, simplifying service and data migration across the cluster with minimal interruption to service and with minimal up-front user configuration.

Various open source projects, such as Hadoop (HDFS), Ceph, and a variety of commercial products are planning to take advantage of Accelio as their high-speed clustering middleware.

### Reliable Transaction Messaging in Trading and Financial Services

Accelio can provide high performance and low latency in a single messaging API with minimal jitter for trading architectures with use cases in electronic exchanges, dark pools, and market data distribution.

Key drivers:

- **High-performance:** low-latency asynchronous API with deterministic jitter
- **Scalability:** connection and resource abstraction with high availability
- **Reliability:** safe end-to-end message delivery and network high-availability/multi-path
- **Maximizing Hardware:** maximizes multi-threaded application performance, utilizes hardware network offloads and OS bypasses
- **Service Location:** dynamic resolution of service objects to endpoints
- **Simple:** Straightforward abstraction API

Key user benefits:

- **Future-proof Infrastructure:** utilize 10, 40, and 100Gb technology seamlessly
- **Simplified Enterprise Messaging:** replaces costly messaging layer
- **Jitter Reduction:** when receiving market data and emitting trades against the dark pool and venues
- **Fast Drop-copy:** for persistence of transactions
- **Simple Fan-out:** one-to-many communications (e.g. a data shredding use case – trading symbols to multiple threads)
- **Features:** allows task cancellation at the API layer

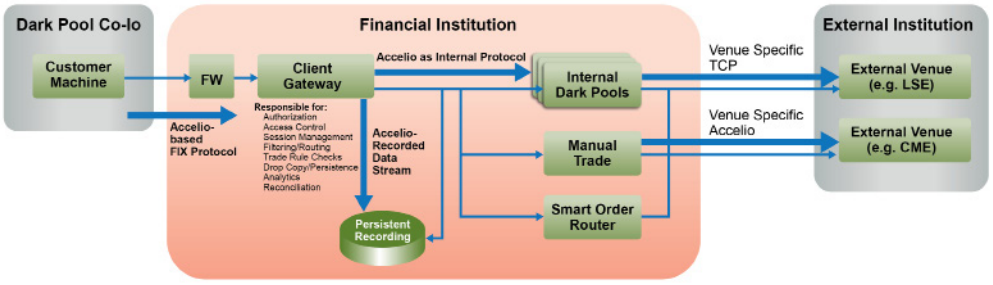


***Figure 8.*** *Using Accelio for Financial Trading*

## Comparing Accelio to Traditional Socket APIs

The following are challenges that developers commonly face with the current socket programming model:

- Most data center protocols (HTTP/REST, FTP, iSCSI, NFS, SQL, RPC, and so on) are transactional with a request/response model, while the sockets API is byte streaming, therefore requiring additional parsing and header/data/transaction processing and layers.
- Socket APIs require heavy protocol processing and copies (due to a lack of message boundaries, constant data copy, and credit/buffer management), and therefore limit the abstraction and offload possibilities.
- The socket connection model cannot address the growing number of CPU cores and NUMA, and applications must use multiple parallel connections and interrupt resources to scale, which adds complexity to the application logic and adds redundant communication threads.
- With sockets, there is no way to steer incoming traffic to the specific/relevant CPU core, and/or use the nearest-by NUMA memory. This leads to additional context switches, CPU lock overhead, and memory coherency overhead.
- Socket APIs do not guarantee reliable delivery to the peer applications (just reception by the peer TCP stack), requiring extra application logic to handle transaction reliability.
- The socket API is limited to a single connection and does not take advantage of multiple network ports or paths. Addressing this requires complex multi-path and connection management layer implementation in the application.
- Important application metrics such as transactions/second, IO latency, and so on cannot be measured at the infrastructure level since there is no correlation between byte streams and messages or between requests and the relevant responses.

These points lead to the clear conclusion that new API semantics are required to take advantage of the new CPU and networking architectures.

The following table compares the traditional TCP socket API with Accelio on various application aspects:

| Aspect | TCP Socket API | Accelio API |
|---|---|---|
| *General* | | |
| API Style | Send/receive bytes | Request/response/send message |
| API Parameters | Bytes + length (application encode/decode messages) | Header, data-in vector, data-out vector, response |
| Peers | 1:1 connection | 1:1, 1:many, many:1 |
| *Scalability and High-Availability* | | |
| Transaction High-Availability | No (requires application logic) | Yes |
| Send Acknowledgement | Reliable *delivery* to remote TCP stack | Reliable acceptance by remote application |

| Aspect | TCP Socket API | Accelio API |
|---|---|---|
| Clustering & Connection Redirection | No (requires application logic) | Yes (redirects connections or requests) |
| *Application Performance* | | |
| Extensive Locking Between CPUs | Yes | No (lockless architecture, dedicated hardware communication resources per CPU) |
| Memory Affinity Between NUMA and CPU Sockets | No, coherency problems and overhead | Yes, allocates memory from local NUMA node |
| Connection and CPU Core Load-balancing | No (requires application logic) | Yes (uses multiple QPs/CQs) |
| Zero-copy | No (requires copy to application buffer) | Yes (uses Accelio or application registered buffers with RDMA) |
| Calls per Transaction | 1 to few | 1 or less (multiple transactions can be pushed in one IO/hardware operation) |
| Interrupt Moderation | Opportunistic (interrupts every 1-n packets) | No interrupts under load (stays in application context when loaded, controlled use of polling) |
| *Other* | | |
| Transaction Monitoring | No (requires application logic) | Yes (latency, msg/sec, etc.) |
| Security (Authentication) | No | Future |

## Summary

Accelio is an innovative API that addresses some of the primary challenges of modern compute, network, and storage systems, such as:

- Efficiently using multiple cores
- Robust scale-out
- Reducing hardware and software stack overhead
- Reducing communication overhead and latency

Accelio allows rapid development of scale-out applications and middleware by providing a robust and flexible communication and messaging layer. It enables consolidation of performance, availability, management, and security features into one implementation that can serve multiple applications. When working and contributing to the Accelio OpenSource community, vendors can focus on the core added values of their applications and can share their implementations and experience with a larger community of developers.

For more information, and to join the developers community, visit www.accelio.org.

Accelio code is hosted on https://github.com/accelio.

Accelio

350 Oakmead Parkway, Suite 100, Sunnyvale, CA 94085
Tel: 408-970-3400 • Fax: 408-970-3403
www.Accelio.org

15-1219WP
Rev 1.1