Benchmarking of FIX Antenna C++ 2.19.0 Linux

Overview

FIX Antenna C++ performance was measured using the sample from the FA package (samples\Benchmark\Latency2). Configuration properties can also be found in the package.

You can reproduce the measurement on your hardware by compiling and executing the sample described above using the following instruction:

How to measure FA performance using the FA package sample

The following instruction provides the steps to measure the FIX Antenna C++ performance using the Benchmark samples on the FA package.

> ⚠ Two Linux machines should be used for the test, one as the Sender Host (server), the other as the Receiver Host (client).

Download the FIX Antenna C++ package, unzip it and go to the \Samples\Benchmark\Latency2 folder.

1. Run the **makefile.ind** to compile all samples.
2. Go to the \Samples\Benchmark\Latency2\bin folder.
   Open the **settings.sh** script and change the IP address from the localhost to the IP of the Server Host in the ServerAdress parameter.

```
#!/bin/bash

LD_LIBRARY_PATH=$LD_LIBRARY_PATH:../../../../lib:/usr/local/lib
export LD_LIBRARY_PATH

export ListenPort=9001
export ListenSSLPort=9000
export ServerAdress=127.0.0.1
export ServerPort=9001

export OnloadServerProfile=latency
export OnloadClientProfile=latency
export TasksetServerCores=16-19
export TasksetClientCores=16-19
```

   On the Server side run the **run.Server.<configuration_name>.sh** script and on the Client side - the **run.Client.<configuration_name>.sh** script. As soon as both of them are executed, the RTT is automatically measured. The **latency.csv** file with results will appear in the same folder.

> ⚠ In the **optimized.hard** configuration the number of cores used by the FIX sessions is directly written in the configuration. Please change it according to your machines. It should be changed in the **settings.sh** file both in the **TasksetServerCores** and **TasksetClient Cores** (please see the picture above) parameters.

> ⚠ In the **optimized.balanced.onload** configuration the Solarflare OpenOnload profile is specified. Please specify your existing configuration or set up the new one in the OpenOnload. It should be changed in the **settings.sh** file both in the **OnloadServerProfile** and **OnloadClientProfile** (please see the picture above) parameters.

Environment

Sender Host:

- Intel(R) Xeon(R) CPU E5-2643 v3 @ 3.40GHz (2 CPU Hyper-Trading Enabled, 24 Cores)
- RAM 128 GB, 2133 MHz
- NIC Solarflare Communications SFC9120 (Firmware-version: 4.2.2.1003 rx1 tx1)

- Linux (CentOS 7.0.1406 kernel 3.10.0-123.el7.x86_64)
- SolarFlare driver version: 4.1.0.6734a

Receiver Host:

- Intel(R) Xeon(R) CPU E5-2687W v3 @ 3.10GHz (2 CPU Hyper-Trading Enabled, 20 Cores)
- RAM 128 GB, 2133 MHz
- NIC Solarflare Communications SFC9120 (Firmware-version: 4.2.2.1003 rx1 tx1)
- Linux (CentOS 7.0.1406 kernel 3.10.0-123.el7.x86_64)
- SolarFlare driver version: 4.1.0.6734a

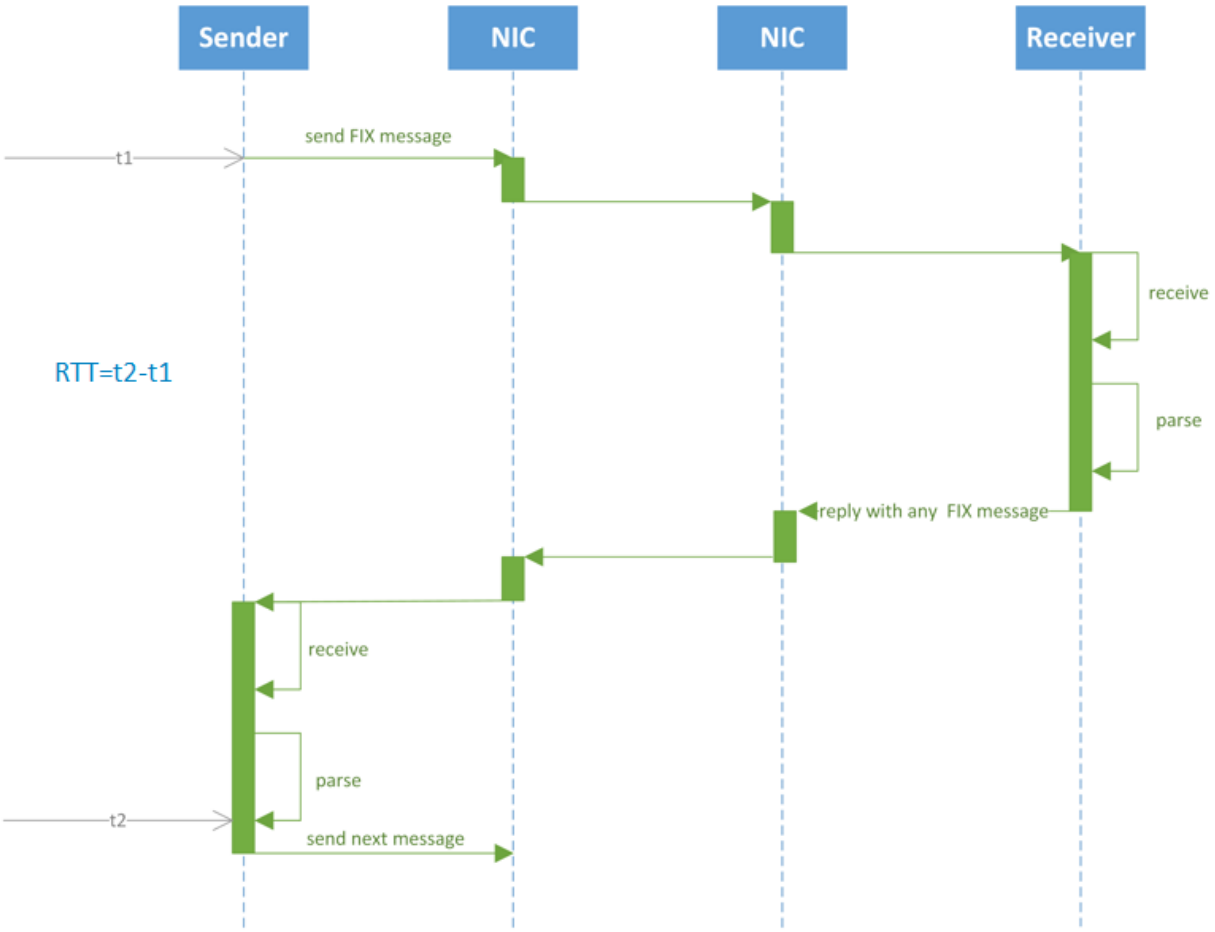
Test scenario

The test scenario is the following:

- Two test servers are connected via 10GB link.
- An Initiator FIX session is established on the Sender Host, an Acceptor FIX session is established on the Receiver Host.
- The Initiator sends the New Order Single (35=D) message to the Acceptor; the Acceptor receives, validates and parses the message and sends the Execution report (35=8) message back to the Initiator.

During the test Round-trip time (RTT) is measured. The first measurement, t1, is made before the message is sent by the Initiator, the second, t2, is made after the received message is parsed by the Initiator.

RTT=t2-t1.



Test configurations

Properties		Default	Aggressive	Balanced	Optimized.hard. spin	Balanced. onload	Optimized. hard
Prepared messages[1]		No	No	No	Yes	No	Yes

Affinity		No	No	Yes	No	Yes	Yes
Thread Pool or Aggressive Mode		Thread Pool (10 Threads)	Aggressive Mode[2]	Aggressive Mode	Aggressive Mode	Aggressive Mode	Aggressive Mode
Message validation parameters	**IgnoreUnknownFields**						
	ProhibitUnknownTags						
	CheckDuplicateTags						
	MessageMustBeValidated						
	VerifyReperatingGroupBounds						
	Validation. CheckRequiredGroupFields						
	Validation.VerifyTagsValues						
	Validation. ProhibitTagsWithoutValue						
Storage type		PersistentMM[3]	PersistentMM	PersistentMM	PersistentMM	PersistentMM	PersistentMM
Solarflare OpenOnload[4]		No	No	No	No	Yes	Yes

[1] Prepared messages - a mechanism that preallocates storage for messages (buffers) and then reuses it.

[2] Aggressive Mode - 2 dedicated threads are allocated for the FIX session: one for sending messages, another - for receiving them.

[3] PersistentMM storage type = a file on the disc.

[4] Solarflare OpenOnload - the kernel bypass technique by Solarflare that is activated within the test.

The following description will help to choose the most relevant configuration of FIX Antenna C++:

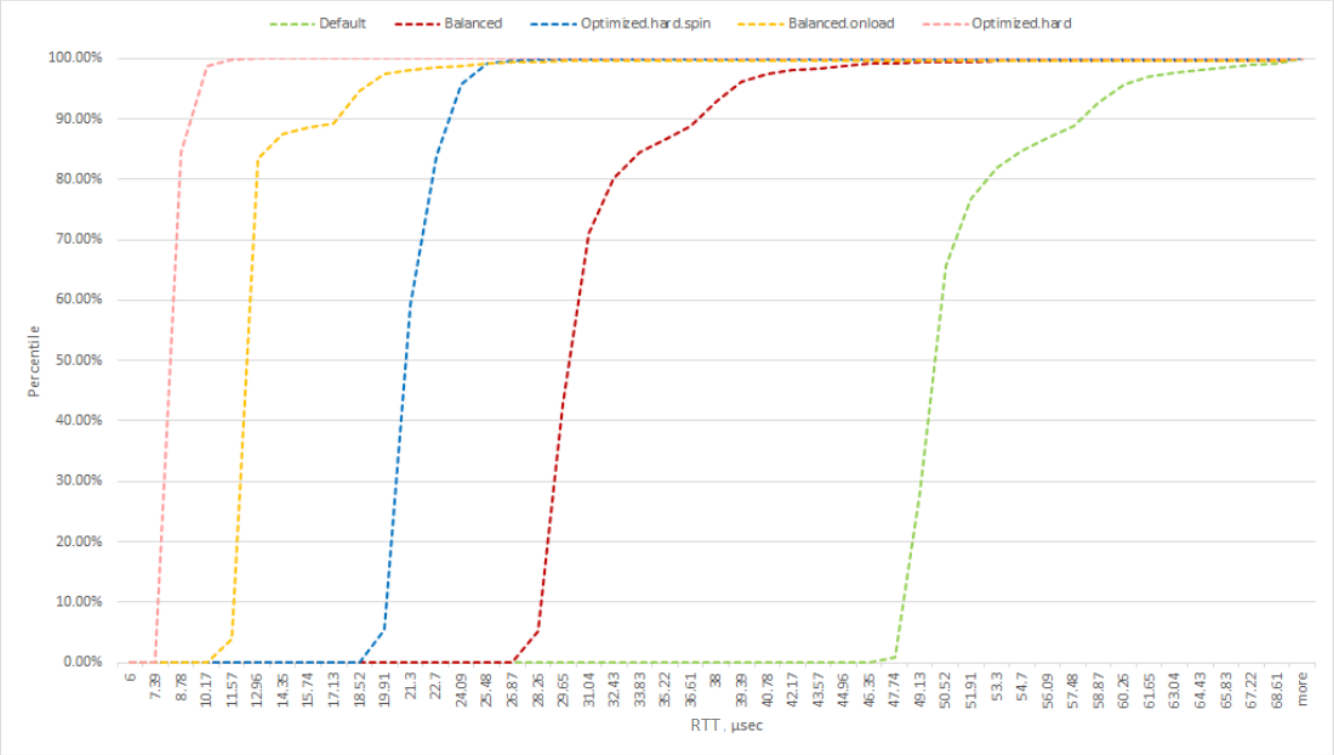
- **Default.** It is the good quick start variant with balanced performance and security. The configuration may be effectively applied both to the Sender Host and the Receiver Host.
- **Aggressive.** It is the optimized version of the default configuration. The aggressive mode is used, so that there are allocated two dedicated threads for each FIX session. The configuration is advised to be used for limited amount of sessions (say, tens or hundreds sessions) and may be effectively applied both to the Sender Host and the Receiver Host.
- **Balanced.** Most validation parameters are switched off to improve performance. The configuration is advised to be used in the controlled environment and is preferable for the Sender Host.
- **Optimized.hard.spin.** The spinning mode is activated in the configuration by the following setting: *Session.Default.AggressiveReceiveDelay = 0,* it forces threads to spin. It is not recommended to use the configuration if the number of threads is significant as it is highly consumable for the CPU. It is advised to be used for the Sender Host.
- **Balanced.onload.** The Solarflare OpenOnload technique activated within the configuration optimizes the TCP stack. However, the configuration may be applied only if there is a Solarflare Network interface controller installed on the machine. It is advised to be used for the Sender Host.
- **Optimized.hard.** For the RTT reduction, the affinity masks are used in the configuration. Also few validation parameters, which are not necessary in the fully controlled environment, were switched off. It is advised to be used for the Sender Host.

Results

FIX Antenna C++ configurations comparison

Configuration	Default	Aggressive	Balanced	Optimized.hard.spin	Balanced.onload	Optimized.hard
RTT (microseconds)						
Min	47,1	46,1	27,3	19,0	11,3	8,3
Max	190,7	218,0	217,5	223,7	199,5	13,2
Average	51,6	51,1	31,5	21,6	13,2	8,7
RTT distribution (percentiles)						
50%	49,8	48,8	29,9	21,0	11,8	8,5

95%	59,9	59,0	38,9	24,0	18,6	9,3
99%	67,1	90,4	45,8	25,3	24,6	10,4



FIX Antenna C++ 2.19.0 vs QuickFIX C++ 1.14.3

The benchmark code was ported to QuickFIX:

Sender Host

```
struct ExperimentTimes
{
    System::u64 putTime_;
    System::u64 processTime_;
};
class Application :
      public FIX::Application,
      public FIX::MessageCracker
{
public:
 void run() {
  times_ = new ExperimentTimes[10000];
  memset( times_, 0, count*sizeof( ExperimentTimes ) );
  // Get a frequency of timer used
  this->timer_frequency = static_cast<double>(HRTimer::frequency());

  FIX42::NewOrderSingle newOrderSingle(
   FIX::ClOrdID( std::to_string(static_cast<long long>(0)) ), FIX::HandlInst( '1' ), FIX::Symbol( "EU" ), FIX::
Side( FIX::Side_BUY ),
   FIX::TransactTime(), FIX::OrdType( FIX::OrdType_LIMIT )
  );
  loggedInSem_.wait();
  this->start_time = System::HRTimer::value();
  for( size_t i = 0; i < 10000; ++i ) {
   // Saving send starting time
   times_[putTimeIndex_].putTime_ = HRTimer::value();
   if(putTimeIndex_ < 10000) {
    ++putTimeIndex_;
```

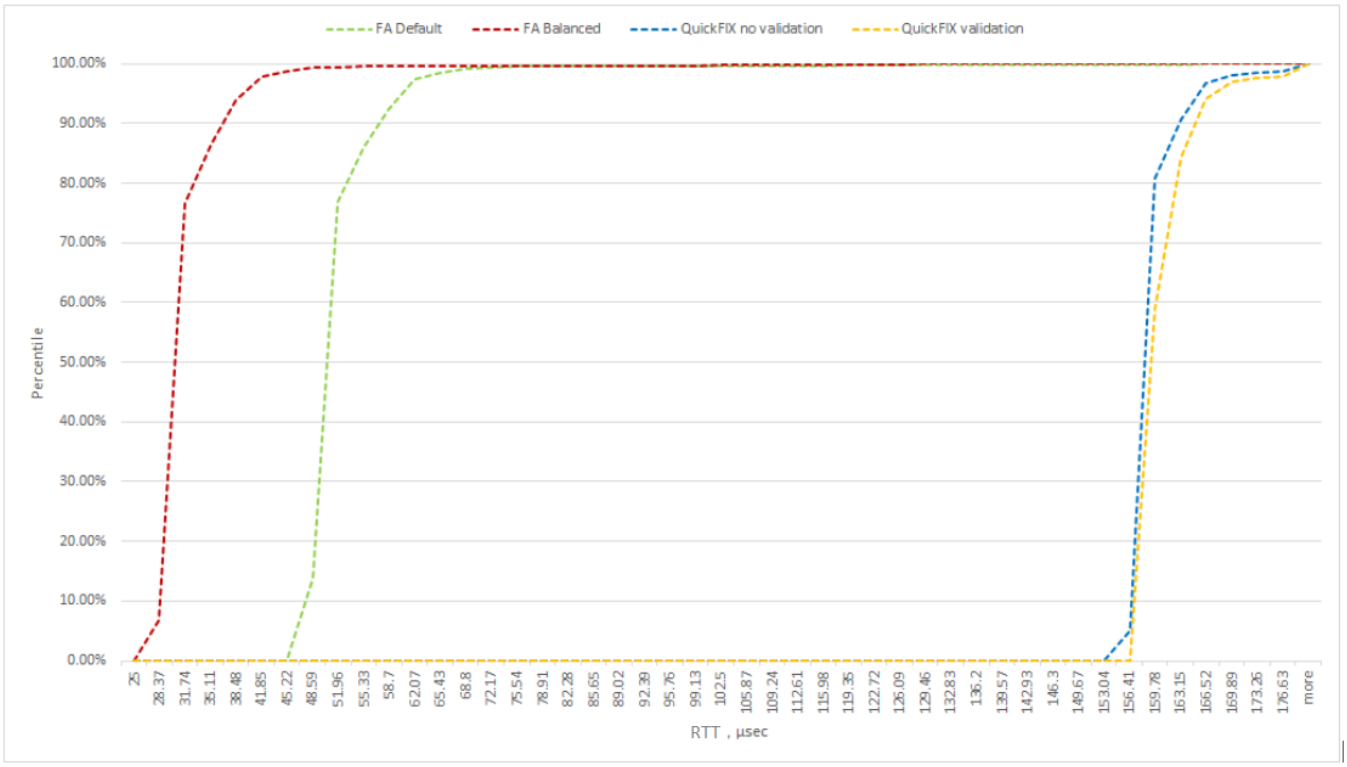
```
   }
   newOrderSingle.set( FIX::ClOrdID( std::to_string(static_cast<long long>(i)) ) );
   newOrderSingle.set( FIX::OrderQty( 1 + i%10 ) );
   newOrderSingle.set( FIX::TimeInForce( FIX::TimeInForce_DAY ) );
   newOrderSingle.set( FIX::Price( 30 ) );
   newOrderSingle.getHeader().setField( FIX::SenderCompID( "INCA" ) );
   newOrderSingle.getHeader().setField( FIX::TargetCompID( "TW" ) );
   FIX::Session::sendToTarget(newOrderSingle);
   // 1K messages per second
   double next_send_time = static_cast<double>(i+1)/1000;
   while(true){
    //Wait for next sending time
    double current_time = static_cast<double>(System::HRTimer::value() - start_time)/timer_frequency;
    double to_sleep = next_send_time - current_time;
    if(to_sleep < 0.0)
     break;
    if(0.025 < to_sleep)
                    Thread::sleep( static_cast<int>(1000.0*0.9*to_sleep) );
   }
  }
 }
private:
 ...
 ExperimentTimes* times_;
 size_t putTimeIndex_;
 size_t processTimeIndex_;

 double timer_frequency;

 Semaphore loggedInSem_;
 ...

 void onLogon( const FIX::SessionID& sessionID ) {
  loggedInSem_.post();
 }

 void fromApp( const FIX::Message& message, const FIX::SessionID& sessionID )
      throw( FIX::FieldNotFound, FIX::IncorrectDataFormat, FIX::IncorrectTagValue, FIX::UnsupportedMessageType
) {
          crack( message, sessionID );
 };
 void onMessage( const FIX42::ExecutionReport&, const FIX::SessionID& ) {
  // Saving message receiving time
  times_[processTimeIndex_].processTime_ = HRTimer::value();
        processTimeIndex_++;
 };
 ...
}
```

Receiver Host

```
class Application :
      public FIX::Application,
      public FIX::MessageCracker
{
 FIX42::ExecutionReport answer;
 int uniqueId;
 ...
 void onMessage( const FIX42::NewOrderSingle&, const FIX::SessionID& ) {
  FIX::SenderCompID senderCompID;
  FIX::TargetCompID targetCompID;
  FIX::ClOrdID clOrdID;
  FIX::Symbol symbol;
  FIX::Side side;
  FIX::OrdType ordType;
  FIX::Price price;
  FIX::OrderQty orderQty;
  FIX::TimeInForce timeInForce( FIX::TimeInForce_DAY );
  message.getHeader().get( senderCompID );
  message.getHeader().get( targetCompID );
  message.get( clOrdID );
  message.get( symbol );
  message.get( side );
  message.get( ordType );
  if ( ordType == FIX::OrdType_LIMIT )
   message.get( price );
  message.get( orderQty );
  message.getFieldIfSet( timeInForce );
  answer.set( clOrdID );
  answer.set( price );
  answer.set( orderQty );
  answer.set( FIX::TransactTime() );
  answer.set( FIX::ExecID(std::to_string(static_cast<long long>(uniqueId)) ) );
  answer.set( FIX::OrderID(std::to_string(static_cast<long long>(uniqueId)) ) );
  uniqueId++;
  answer.set( FIX::LeavesQty(orderQty) );
  FIX::Session::sendToTarget(answer);
 };
 ...
}
```

Results can be compared in the following table:

Configuration	FIXAntenna default	FIXAntenna balanced	QuickFIX no validation	QuickFIX validation
RTT (microseconds)				
Min	47,1	27,3	155,0	157
Max	190,1	217,5	371,5	377,8
Average	51,6	31,5	158,9	161,8
RTT distribution (Percentiles)				
50%	49,8	29,9	157,6	159,5
95%	59,9	38,9	165,0	167,1
99%	67,1	45,9	177,4	183,5


FA Default
FA Balanced
QuickFIX no validation
QuickFIX validation
100.00%
90.00%
80.00%
70.00%
60.00%
50.00%
40.00%
30.00%
20.00%
10.00%
0.00%
Percentile
25
28.37
31.74
35.11
38.48
41.85
45.22
48.59
51.96
55.33
58.7
62.07
65.43
68.8
72.17
75.54
78.91
82.28
85.65
89.02
92.39
95.76
99.13
102.5
105.87
109.24
112.61
115.98
119.35
122.72
126.09
129.46
132.83
136.2
139.57
142.93
146.3
149.67
153.04
156.41
159.78
163.15
166.52
169.89
173.26
176.63
more
RTT , µsec