FIX Antenna HFT Benchmarks

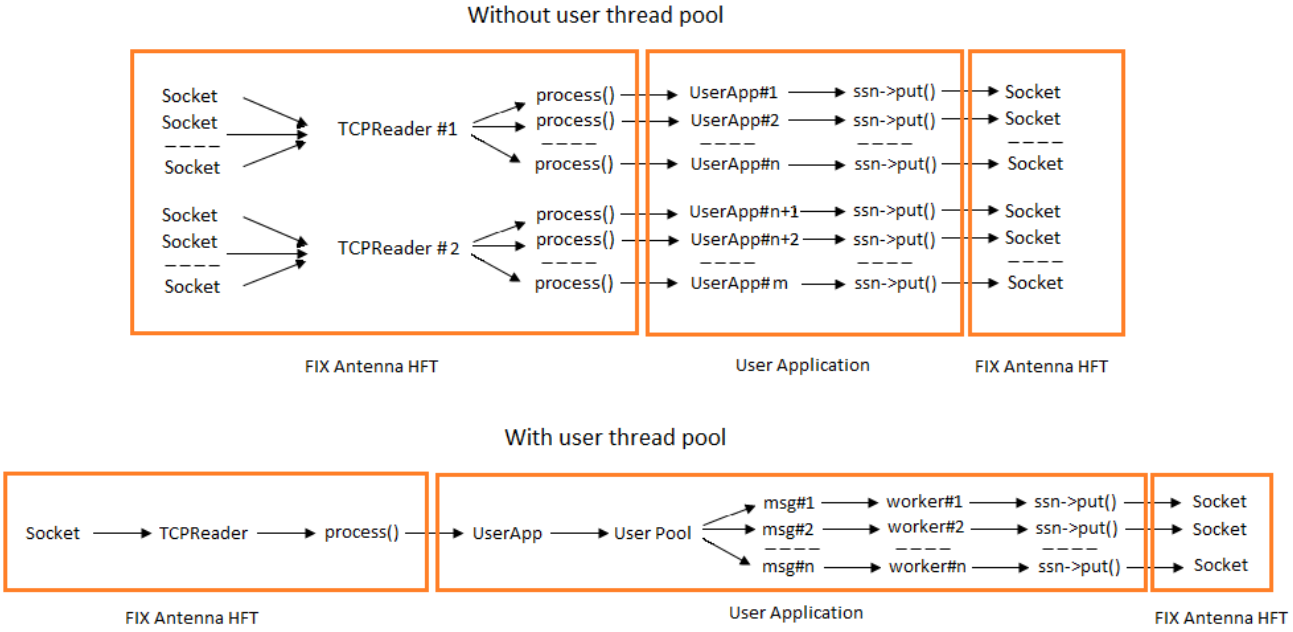
Overview

There are two main patterns which can be used when processing incoming messages in FIX Antenna HFT:

1. Processing messages in the context of the FIX Antenna's receiver thread, and
2. Processing messages in the user's thread.

The first one is used when message processing or routing does not take significant amount of time. In this case messages could be routed directly from the **process()** callback method. This assures the lowest latency possible.
If the number of incoming sessions is big, it is advised to set **TCPDispatcher.NumberOfWorkers = 3**, which means 3 dedicated threads for running **epoll_wait** calls. Also, it is recommended to either set **TCPDispatcher.IncomingConnectionOnloadStackAffinity = false** or specify several listen ports - this will assure even load distribution from all acceptors.

The second one (called a **thread pool**) is used when message processing or routing takes significant amount of time. In this scenario, the client code should create a thread pool (using **Utils::ThreadsPool** class) inside the application and pass the messages pointer to the internal queue in the process() callback method in order the thread pool to start processing messages.
The above considerations are illustrated on the diagram below:



Actually, there are many sessions in the 'With user thread pool' diagram, each of them works as the shown one.

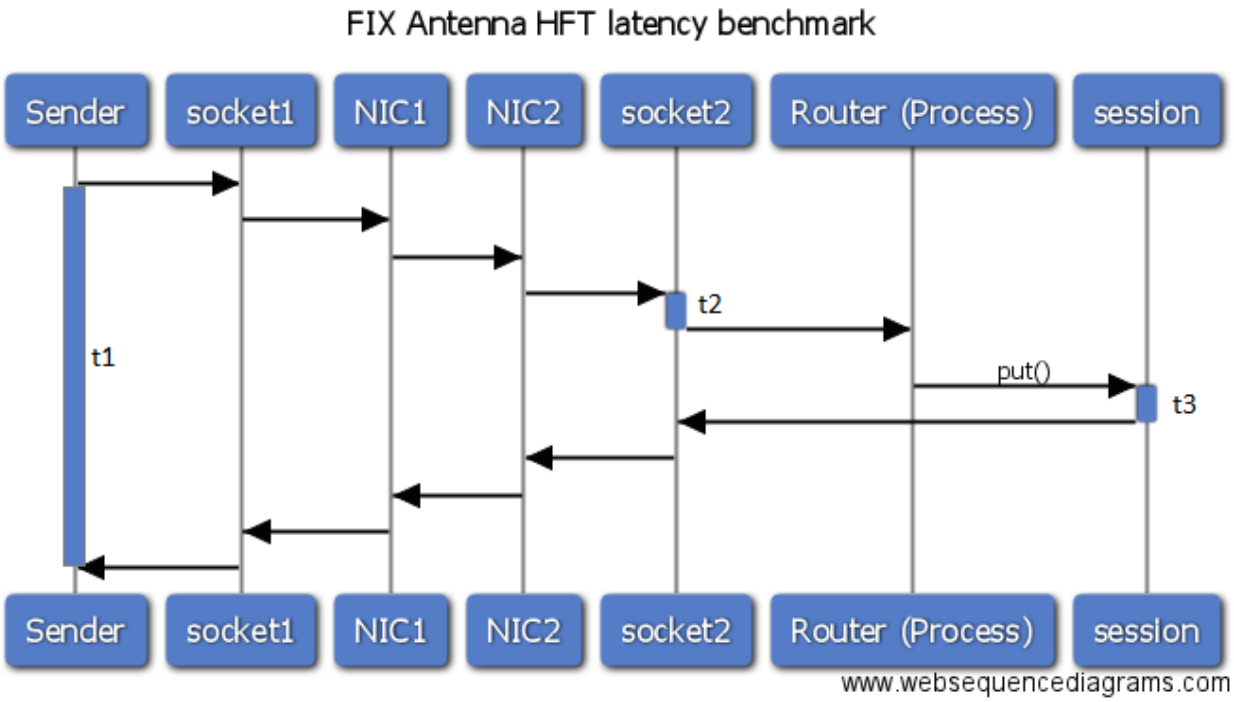
Test scenario

The test scenario is the following:

- Two test servers are connected via LAN.
- The Sender application is launched on the first server, the Router application is launched on the second server.
- The Sender establishes several FIX sessions to the Router and sends messages to the Receiver in all sessions simultaneously.
- The Receiver, when receives a message from any of the established sessions, then sends this message to randomly picked up session back to the Sender.

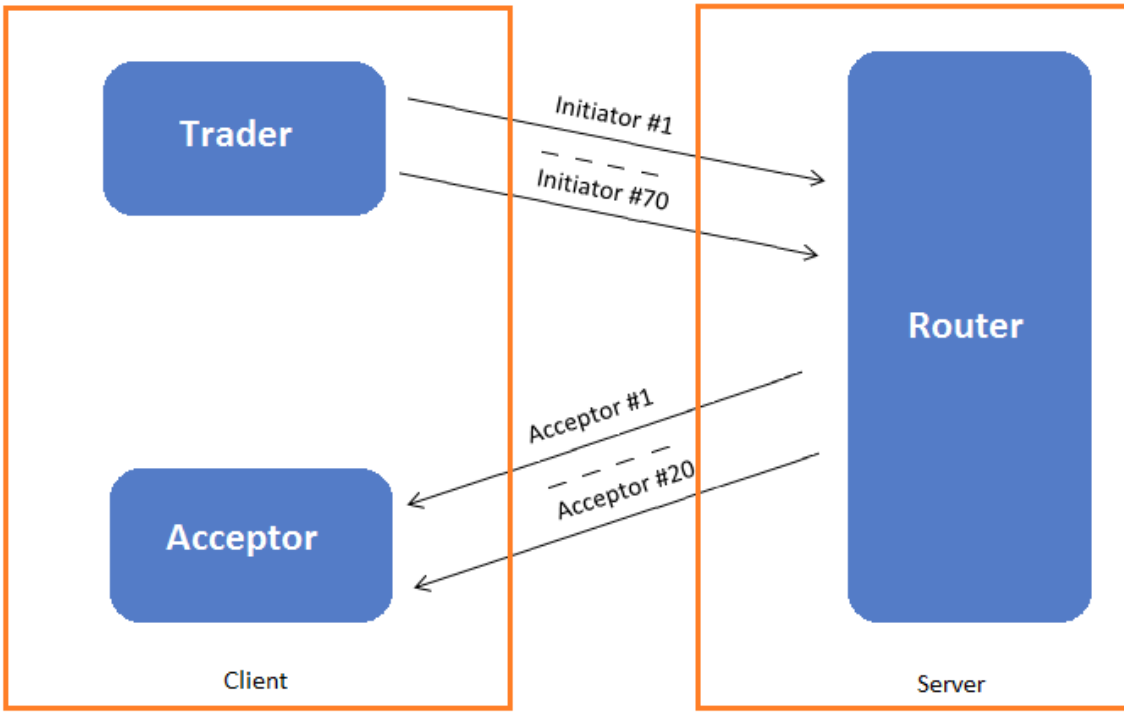
The following parameters are measured during the test:

- Round-trip time (RTT);
- Socket to the process() callback latency on the simple router;
- The put() callback to socket latency on the simple router.



t1-round-trip time
t2-socket to process() time
t3-put() to socket time

The test configuration schema (70 Sender (Initiator) sessions - 20 Receiver (Acceptor) sessions) is the following:



Environment

Machine 1:

CPU: 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
HDD

Centos 7, 3.10.0-123
SolarFlare driver version: 4.1.0.6734a
firmware-version: 4.2.2.1003 rx1 tx1

Machine 2:

CPU: 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
HDD

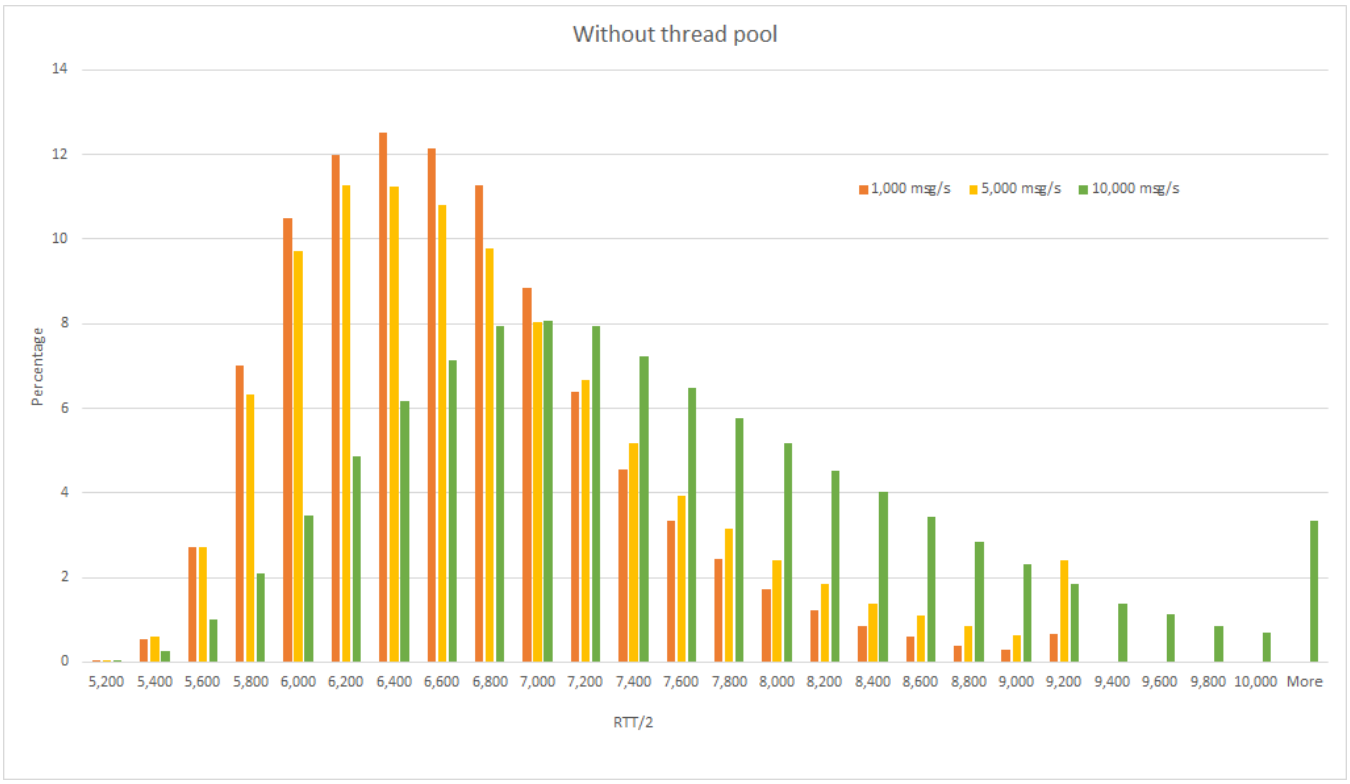
CentOS 7, 3.10.0-123
SolarFlare driver version: 4.1.0.6734a
firmware-version: 4.2.2.1003 rx1 tx1

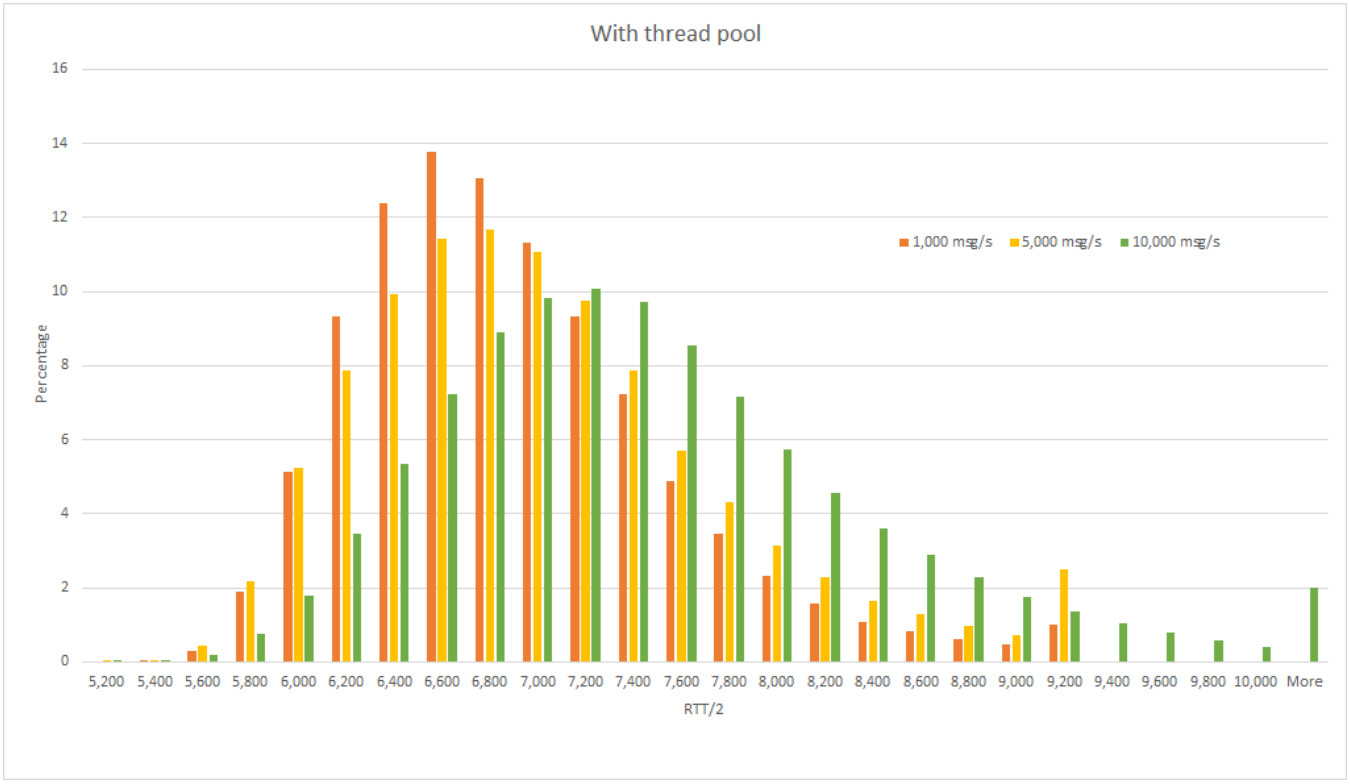
Results

Test results for different number of Sender sessions and Receiver threads (the 50th percentile values) are presented in the table below.

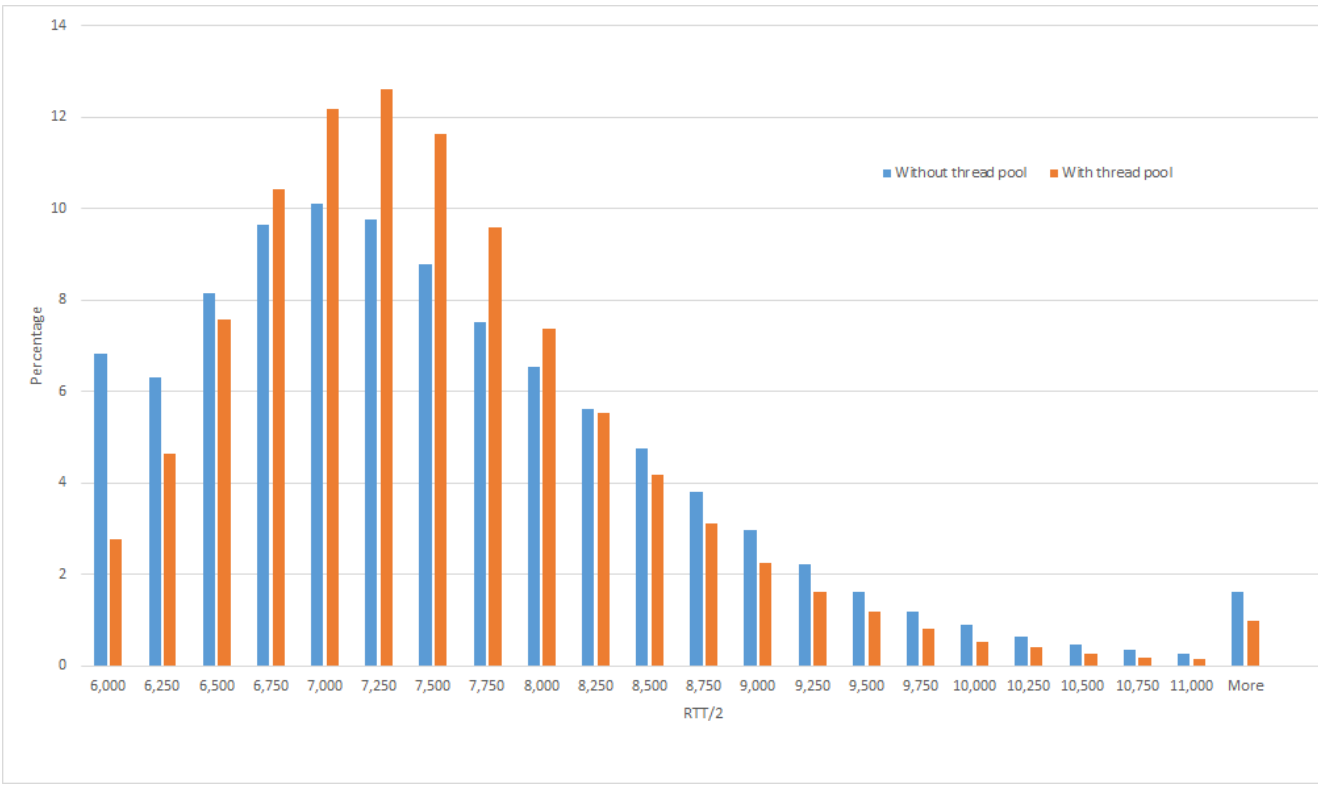
Number of senders (sources)	Message rates per sender, msg/s	Number of receivers (UserApps) (options to route to)	Message rates total for all senders, msg/s	Without thread pool			With thread pool		
				RTT /2, ns	Socket to process() latency, ns	put() to socket latency, ns	RTT /2, ns	Socket to process() latency, ns	put() to socket latency, ns
1	1,000	10	1,000	4,788	385	591	5,147	780	678
1	5,000	10	5,000	4,688	372	556	5,210		
1	10,000	10	10,000	4,644	391	574	5,196		
5	1,000	10	5,000	5,035	401	579	5,346	788	662
5	5,000	10	25,000	5,024	419	552	5,508		
5	10,000	10	50,000	4,935	410	558	5,301		
20	1,000	20	20,000	5,517	-	-	5,920	843	684
20	5,000	20	100,000	5,597	-	-	5,865		
20	10,000	20	200,000	5,504	-	-	5,799		
70	1,000	20	70,000	6,479	473	620	6,712	882	680
70	5,000	20	350,000	6,550	436	550	6,835		
70	10,000	20	700,000	7,213	-	-	7,243		

The histograms below show the distribution of the RTT/2 for 1,000; 5,000 and 10,000 msg/s message rates for configuration with 70 senders and 20 receivers without thread pool and with thread pool.





The following histogram compares the distribution of RTT/2 for 10,000 msg/s message rate for configuration with 70 senders and 20 receivers without thread pool and with thread pool.


14
12
10
8
6
4
2
0
Percentage
Without thread pool
With thread pool
6,000
6,250
6,500
6,750
7,000
7,250
7,500
7,750
8,000
8,250
8,500
8,750
9,000
9,250
9,500
9,750
10,000
10,250
10,500
10,750
11,000
More
RTT/2