

# FAST Protocol (SM)

**FIX**PROTOCOL  
INDUSTRY-DRIVEN MESSAGING STANDARD™

## Market Data Optimized for High Performance



**Matt Simpson, CME, FIX Global Tech Committee  
FIA Expo 2005**

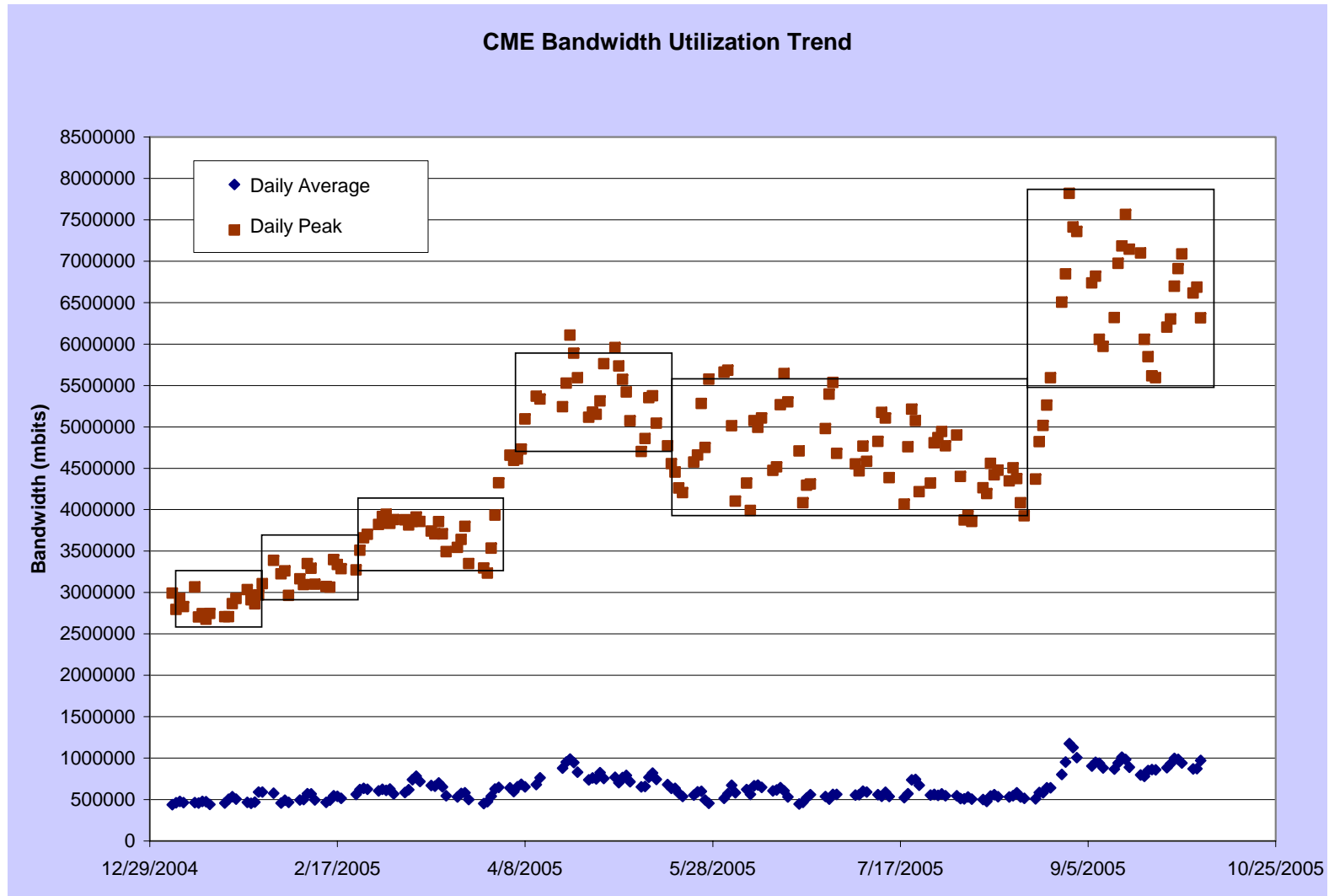


## What is High Performance Market Data?

- Highly Scalable
  - Peaks and spikes are handled without disruption to the market
  - Performance does not degrade as load increases – *it improves*
- Low Latency
  - Time from Sender to Receiver is reduced!
  - Latency Decreases as Compression Increases!
- Low Bandwidth Utilization
  - Allows existing bandwidth to be leveraged
- Low Cost
  - Open, Standard, Repeatable



# CME Market Data Growth in 2005





## What is the FAST Protocol?

FAST stands for FIX Adapted for STreaming

The FAST Protocol...

- Is a highly efficient means of compressing data and reducing latency
- Is especially efficient on data that has repeating patterns and occurs at high frequency as is the case for market data
- Is an encoding algorithm which reduces the size of a data stream through Field Encoding and Serialization
- Works equally well on FIX tag/value or Proprietary Feeds
- Is Template driven – field sequence, data types and encoding operations are conveyed through the template



## How Does FAST Work?

FAST consists of 3 logical layers:

Implicit Tagging +  
Field Encoding +  
Serialization

---

Highly Optimized Feed

Let's take a look at each of these



## FAST Protocol Encode Structure

### Implicit Tagging

- Fixed length data becomes variable length – leading zeros and trailing spaces are dropped
- Delimiters become necessary to separate fields
- Tag Numbers are dropped
- Templates are important in identifying and sequencing fields in a message
- An abstract concept useful in visualizing the structure of the encoded data – is not physically implemented

### Field Encoding

- Eliminates data redundancy by leveraging data affinities between recurring fields in messages and repeating groups
- Fields may be defaulted, copied, incremented or delta valued from a prior occurrence
- Field Encoding operators are specified in the template and applied at the encode level



# FAST Protocol Encode Structure

## Serialization

- **Binary-based Representation**
  - Serialization involves the transformation of the data to a binary-based representation
- **Continuation Bit**
  - Further efficiency gains are realized through bit-level field delimitation known as the “continuation bit”
- **Presence Map**
  - Bit map which indicates the presence or absence of fields in the message. Fields which are *null* due to having been “copied away” can be represented by a bit in the presence map indicating that the field is absent.



## FAST Protocol Latency and Processing Overhead

How FAST is FAST? – Very

Low Processing Latency +  
Reduced Transfer Latency

---

Accelerated Message Delivery

- Extremely Efficient at the CPU-level
  - FAST's binary-like format allows extremely efficient processing within a CPU
- Compression decreases the amount of data that has to be transferred and therefore accelerates data delivery
- The result is that Transfer Latency is decreased
- Fast enough to keep up with huge amounts of data 50 Mbit/s



## FAST is Transport Flexible

### Transport Compatible

- FAST leverages the properties of the transport to optimize performance
- FAST works with either **Broadcast** (multicast) or **Point-to-point** (TCP) transport model
- For Multicast, a single packet defines the **scope of a FAST “transaction”** - FAST never spans multiple packets
- Alternatively, data can be pulled in and decoded **byte-for-byte** or in **packet-like chunks**

### FAST on the Wire

- FAST recognizes a special “template byte” at the beginning of each message which links it to a template for encoding and decoding of the message
- If a template byte is not provided FAST will use the standard Message Type (tag 35) as the template identifier
- For byte-oriented transports, FAST supports a **message length** field or “**start-of-frame**” delimiter which can be used to read the data in blocks



## Development of the FAST Protocol

### Inception

- Development of Specification
- Review of other compression approaches; ASN.1, Huffman, LZ
- Preliminary analysis and testing to optimize encoding

### Elaboration

- Finalize Field Encoding and SERDES specifications
- Mapping of native feeds to FIX
- Design POC Test Framework
- Convergence of early approaches



## Development of the FAST Protocol

### Construction

- Build FAST Components and Test Harness
- Build Metrics and Reporting Components
- Build Test Framework

### Transition

- Feeds are redundantly and reciprocally tested
- Version control of code-base
- Verification – no data loss, no data gain, every byte identical
- Metrics generated, compiled, and cross-checked
- Results Published



## FAST Protocol Proof of Concept



arca|ex®



**Microsoft**





## Rolf and Dan at work





## FAST Protocol Proof of Concept

### Phase1A Summarized Native Results

Exchange Feed	Native Message Size	FAST Message Size	FAST Protocol – Avg Compression Rate	FAST Protocol – Peak Compression Rate	FAST Protocol – Messages per second
<b>ARCA ArcaBook</b>	<b>38.0</b>	<b>9.0</b>	<b>76.2%</b>	<b>80.1%</b>	<b>714,000</b>
<b>OPRA</b>	<b>67.0</b>	<b>16.8</b>	<b>75.9%</b>	<b>81.1%</b>	<b>555,000</b>
<b>CME Globex</b>	<b>195.9</b>	<b>38.6</b>	<b>80.3%</b>	<b>82.0%</b>	<b>294,000</b>
<b>NOREX</b>	<b>113.0</b>	<b>17.2</b>	<b>84.7%</b>	<b>89.8%</b>	<b>588,000</b>

### Phase1B Summarized FIX Results (preliminary)

Exchange Feed	FIX Message Size	FIX FAST Message Size	FAST Protocol – Avg Compression Rate	FAST Protocol – Peak Compression Rate	FAST Protocol – Messages per second
<b>ARCA ArcaBook</b>	<b>121.9</b>	<b>10.5</b>	<b>91.3%</b>	<b>94.1%</b>	<b>667,000</b>
<b>OPRA</b>	<b>177.0</b>	<b>20.8</b>	<b>88.2%</b>	<b>90.8%</b>	<b>434,000</b>
<b>CME Globex</b>	<b>241.4</b>	<b>29.0</b>	<b>88.0%</b>	<b>88.9%</b>	<b>333,000</b>
<b>LSE</b>	<b>157.7</b>	<b>54.4</b>	<b>65.5%</b>	<b>81.5%</b>	<b>149,000</b>



## Test Phases

- Phase1A addressed Native Feeds
  - ARCA, OPRA, CME and NorEx as test subjects
  - Top-of book + Trades (OPRA)
  - Full Depth Order Book (Archipelago Exchange)
  - Aggregate Depth Book + Trades (CME)
  - Full Depth Order Book + Trades (LSE/NorEx)
- Phase1B addressed FIX-adapted Feeds
  - ARCA, OPRA, CME and LSE as test subjects
  - FIX Market Data Snapshot Full Refresh (35=W)
  - FIX Market Data Incremental Refresh (35=X)
- Phase2 will continue testing exchange feeds
  - SWX, SGX, CBOE
  - ZLIB Comparisons



## POC FIX Input

### FIX Market Data Incremental Refresh Message

- CME Book Update Message
- 4 Price Levels Specified
- Each Price Level contains repeating information

```
8=FIX.4.4|9=348|34=36|35=X|49=CME|268=4|  
279=1|269=0|107=GE:BF Z5-H6-M6|270=2.5|271=533|272=20050725|273=072000|276=K|  
279=1|269=1|107=GE:BF Z5-H6-M6|270=3.5|271=211|272=20050725|273=072000|276=K|  
279=1|269=0|107=GE:BF Z5-H6-M6|270=2|271=33131|272=20050725|273=072000|276=K|  
279=1|269=1|107=GE:BF Z5-H6-M6|270=4|271=15345|272=20050725|273=072000|276=K|  
10=080|
```

- ARCA “Add Order” Message

```
8=FIX.4.4|9=116|34=23|35=X|49=ARCA|52=04:00:00.878|268=1|  
55=AOLA|299=ARCA|37=7|269=1|279=0|270=0.3|271=16000|275=P|  
10=234|
```

# POC Summary Results

## ARCA FIX FAST Performance

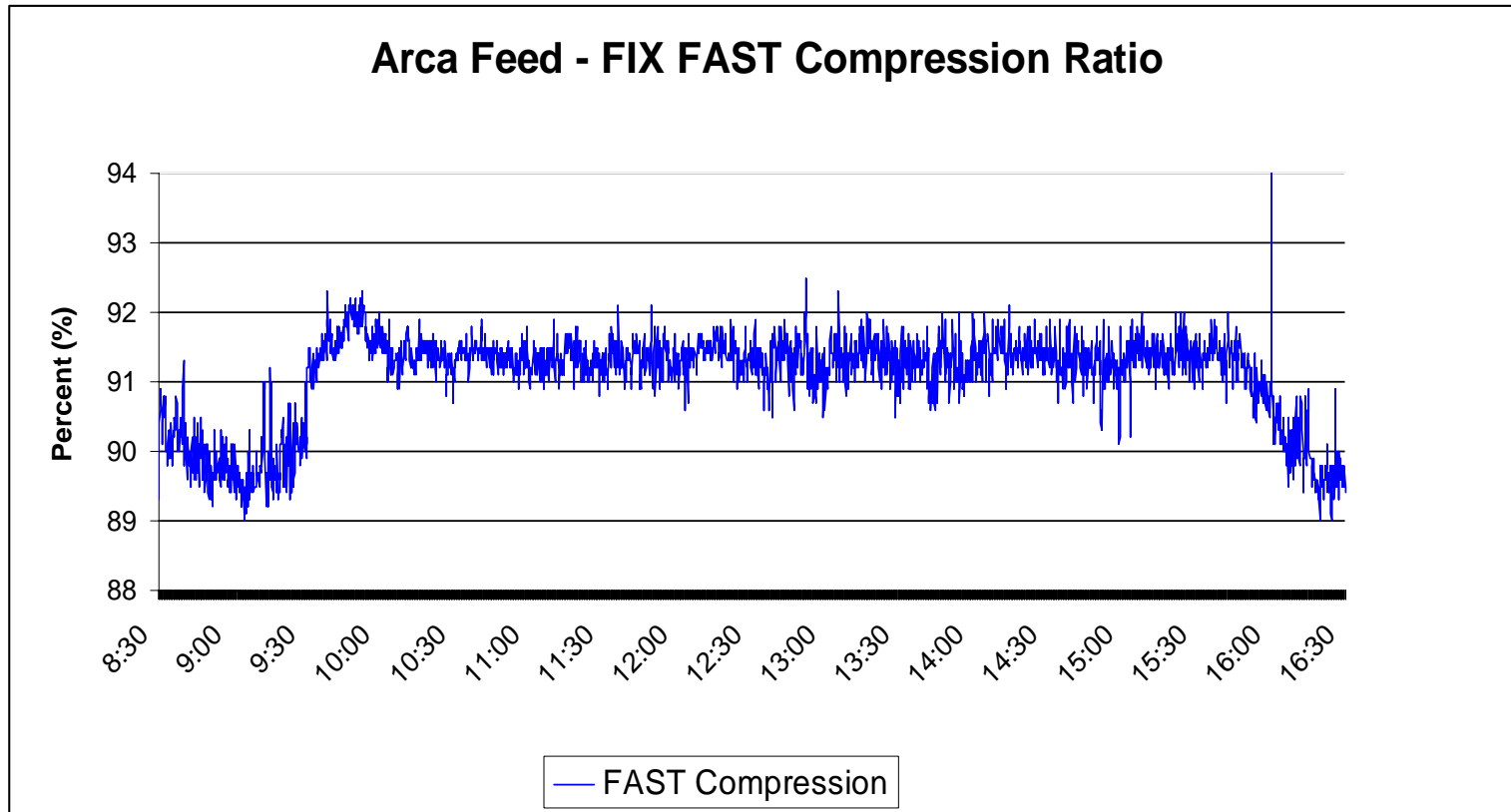


Test Category	Result
Trade Date	May 11 <sup>th</sup> , 2005
Maximum Compaction Factor (Percent/Ratio)	94.1% / 5.03
Average Compaction Factor (Percent/Ratio)	91.37% / 4.20
Minimum Compaction Factor (Percent/Ratio)	89.4% / 3.24
Frame Size	1400 bytes
Total number of messages	77,257,770
Average FIX message size before compression	121.93 bytes
Average FIX message size after compression	10.52 bytes
Total number of bytes before compression	9,419,817,905 bytes
Total number of bytes after compression	812,906,076 bytes
Total number of frames per test	577,931 frames
Average number of messages per frame	133.68 messages
Total CPU Utilization	424 CPU seconds
Average CPU Utilization per message	1.5 microseconds





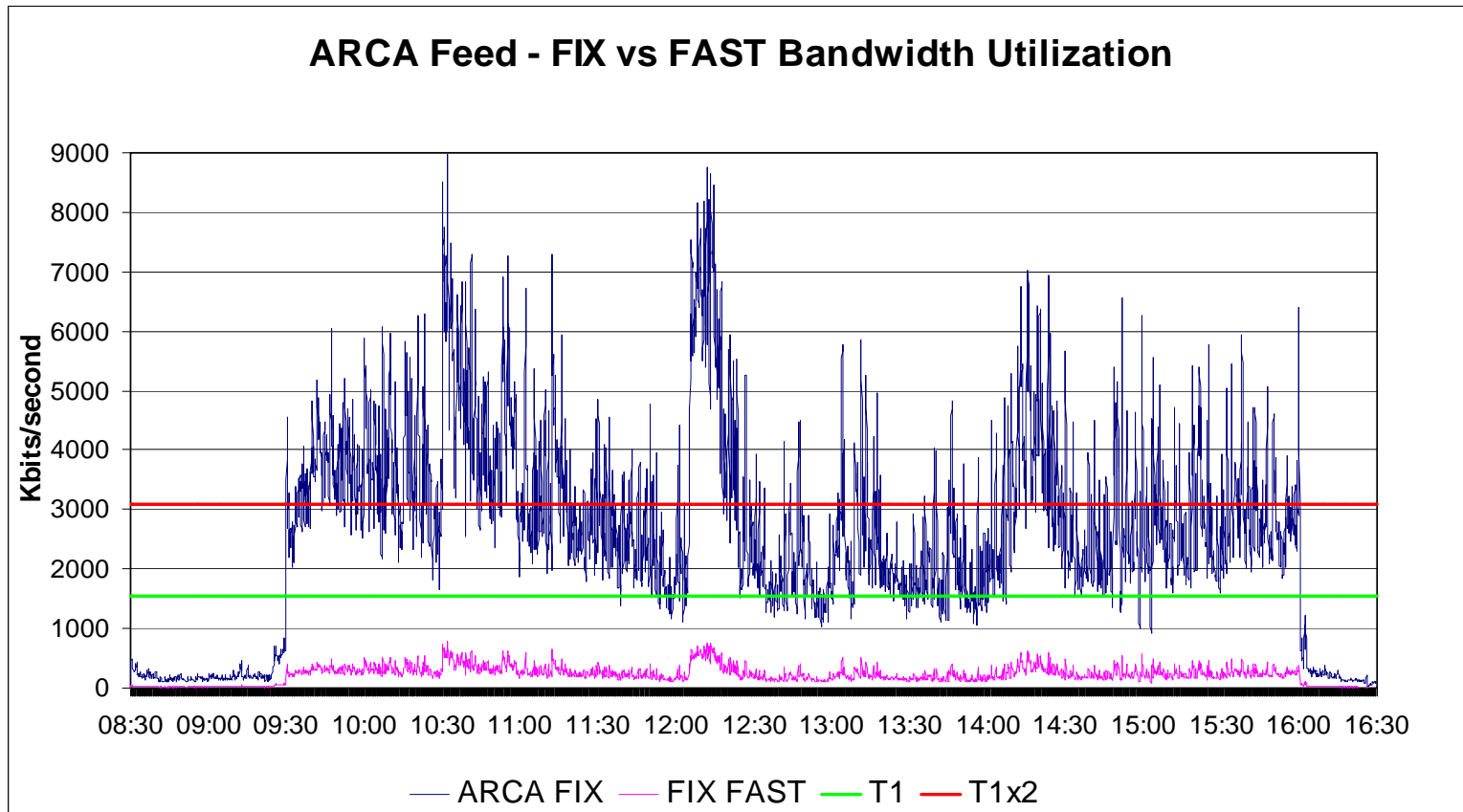
## ARCA FIX FAST Performance FAST Protocol Using FIX - ARCA Compression Ratio Performance





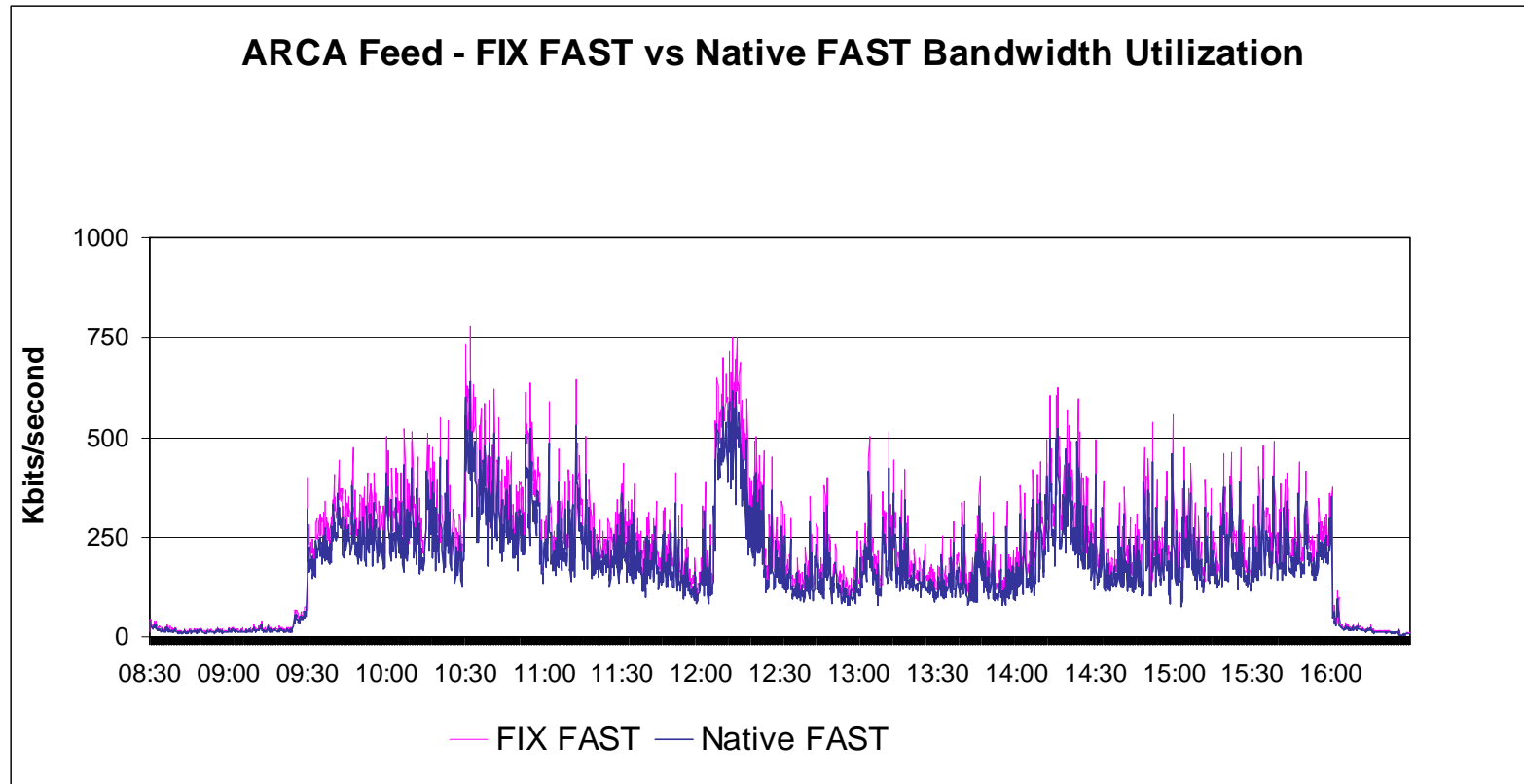
## ARCA FIX FAST Performance

### FAST Protocol vs. FIX - ARCA Bandwidth Utilization





## ARCA FIX FAST Performance FIX FAST vs. Native FAST – ARCA Bandwidth Utilization



# POC Summary Results

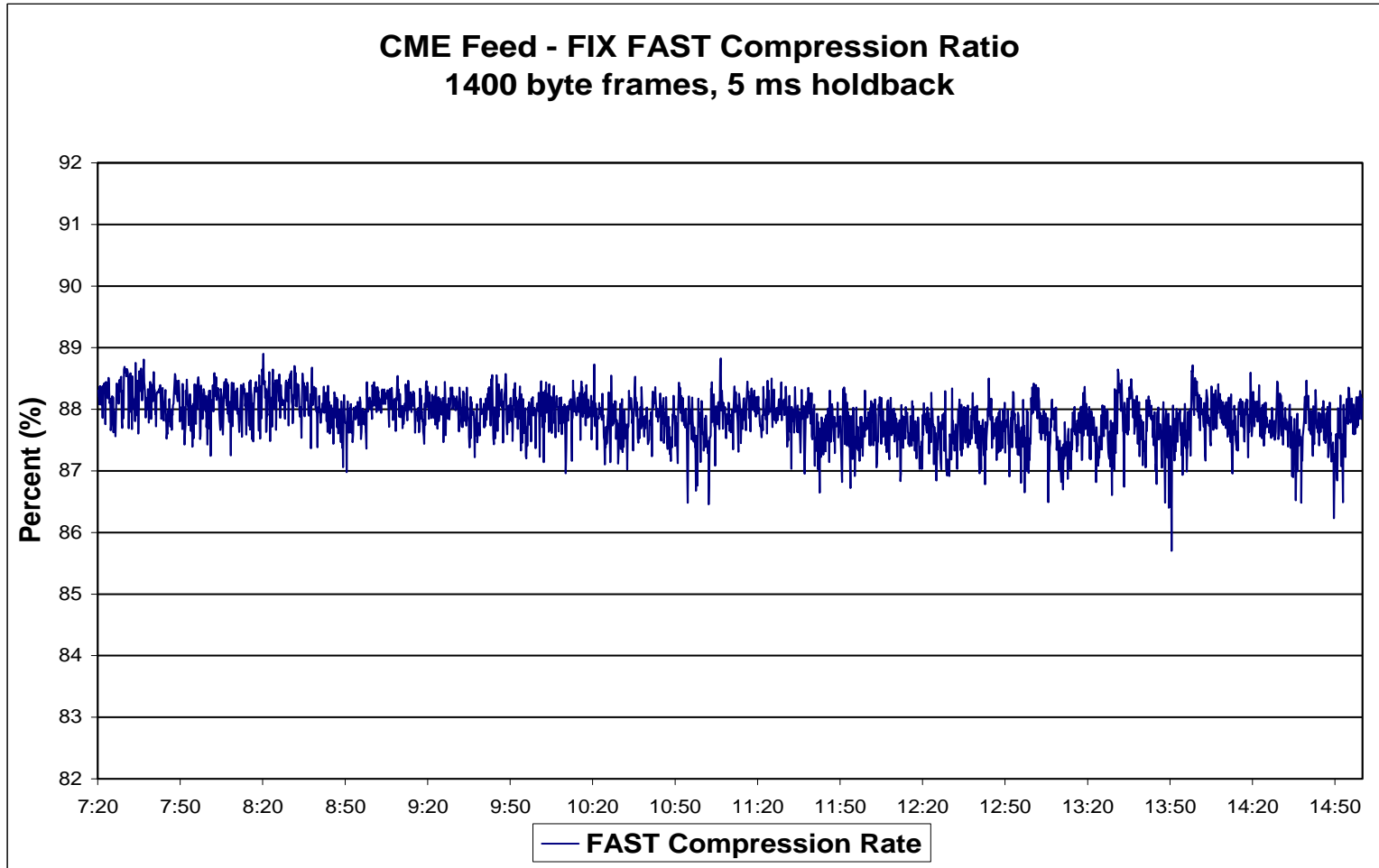
## CME FIX FAST Performance



Test Category	Result
Trade Date	July 25, 2005
Maximum Compaction Factor (Percent/Ratio)	88.90% / 9.00
Average Compaction Factor (Percent/Ratio)	88.00% / 8.33
Minimum Compaction Factor (Percent/Ratio)	86.78% / 7.32
Frame Size/Holdback	1400 bytes / 5ms
Total number of messages	7,357,984
Average FIX message size before compression	241.41 bytes
Average FIX message size after compression	29.03 bytes
Total number of bytes before compression	1,441,663,338 bytes
Total number of bytes after compression	283,723,183 bytes
Total number of frames per test	3642715
Average number of messages per frame	3.0
Total CPU Utilization	23.2 seconds
Average CPU Utilization per message	3.0 microseconds

# CME FIX FAST Performance

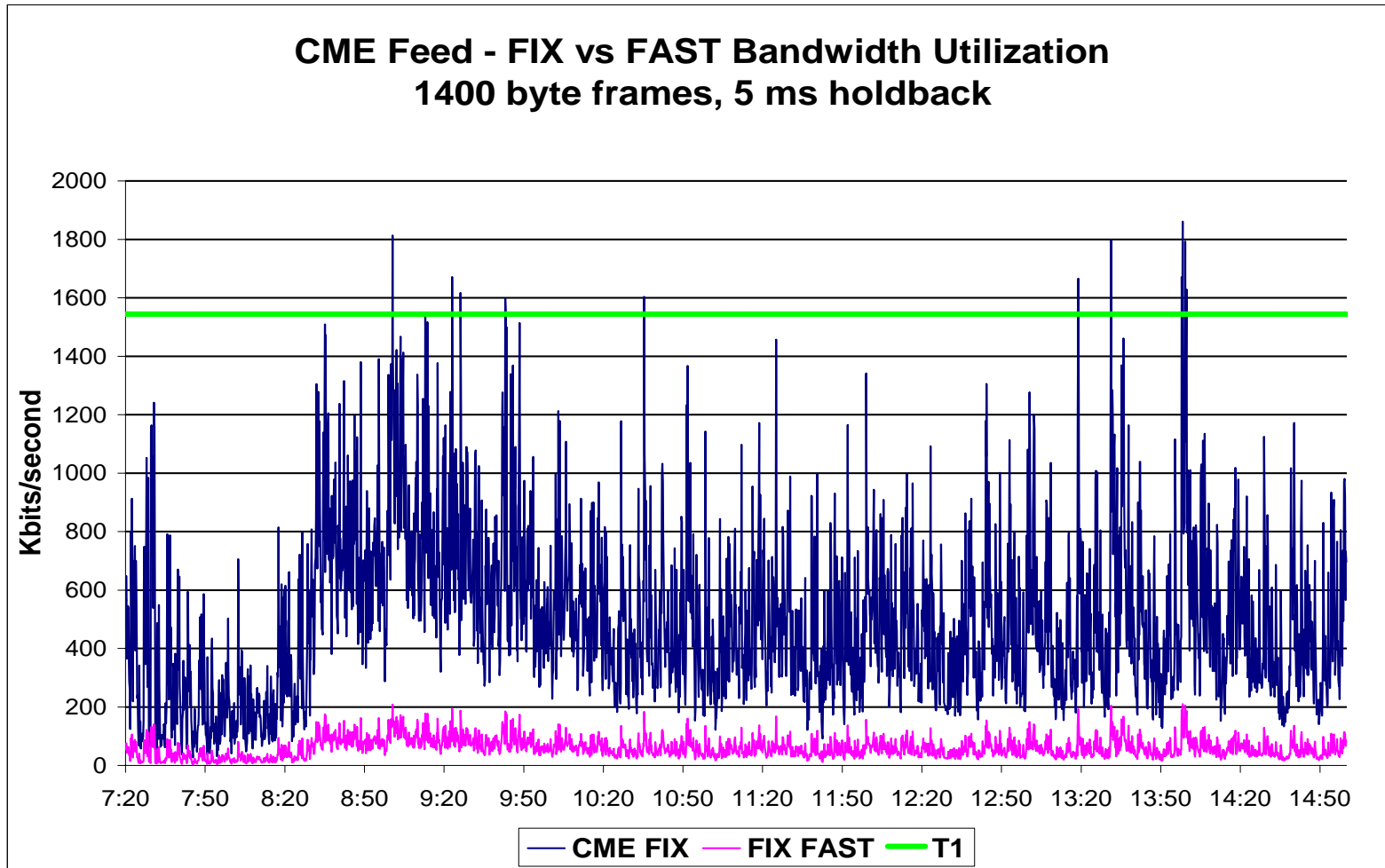
## FAST Protocol Using FIX - CME Compression Ratio Performance





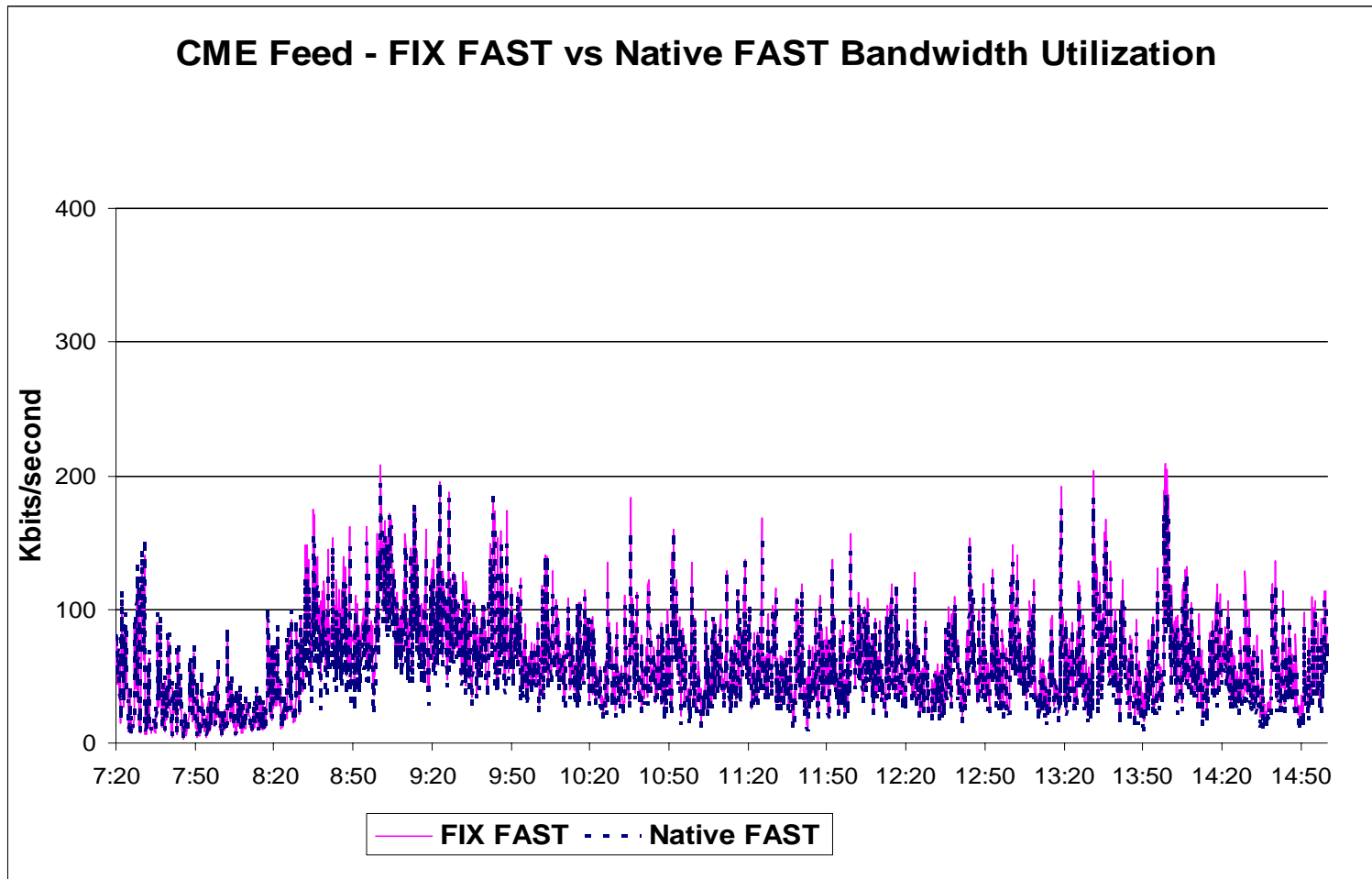
## CME FIX FAST Performance

### FAST Protocol vs. FIX - CME Bandwidth Utilization





## CME FIX FAST Performance FIX FAST vs. Native FAST – CME Bandwidth Utilization



# POC Summary Results

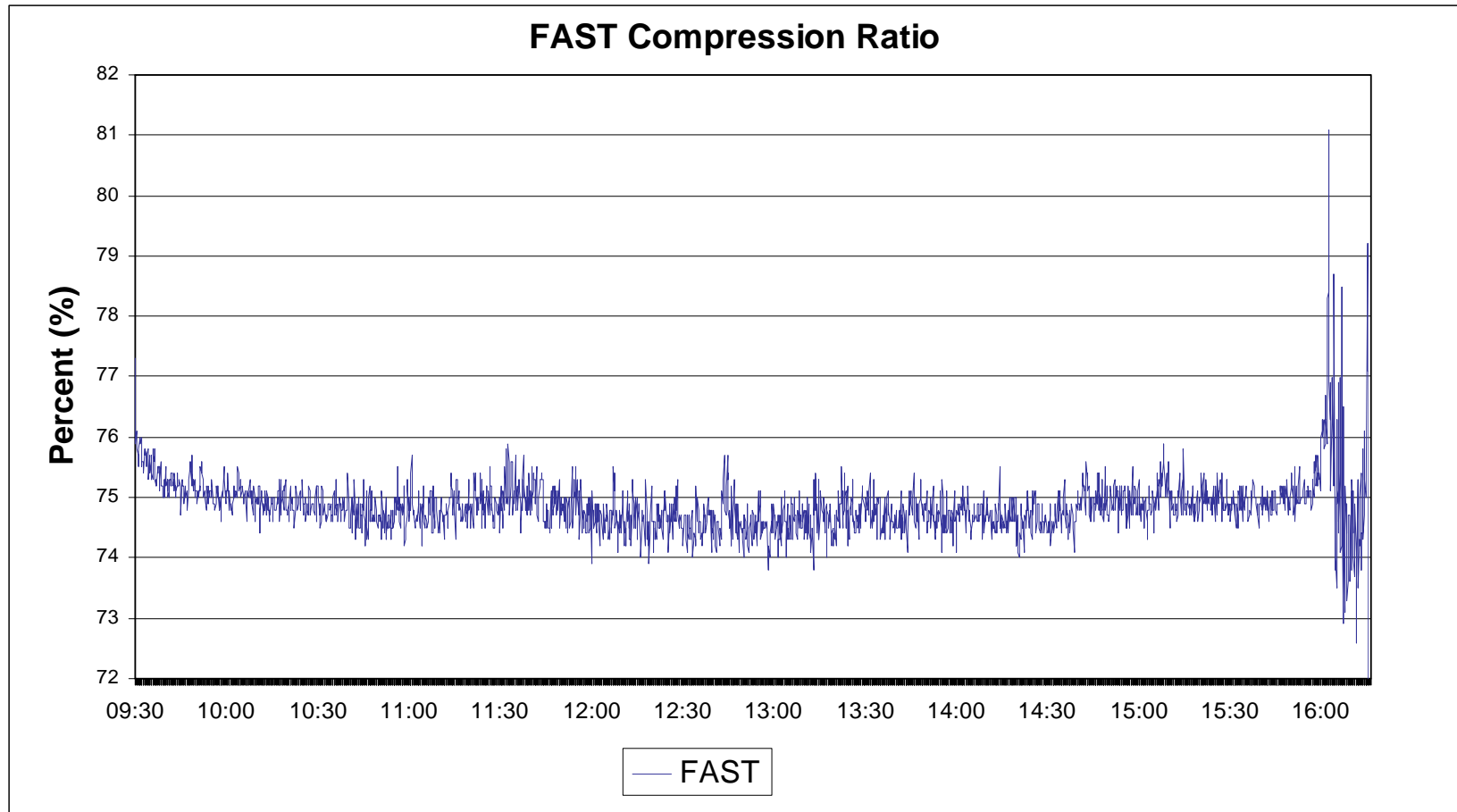
## OPRA Native FAST Performance



Test Category	Result
Trade Date	July 19, 2005
Maximum Compaction Factor (Percent/Ratio)	81.1% / 5.29
Average Compaction Factor (Percent/Ratio)	75.90% / 3.98
Minimum Compaction Factor (Percent/Ratio)	72.9% / 3.69
Frame Size	1000 bytes
Total number of messages	67,553,409
Average message size before compression	67.00 bytes
Average message size after compression	16.81 bytes
Total number of bytes before compression	4,526,329,049 bytes
Total number of bytes after compression	1,135,908,467 bytes
Total number of frames per test	1,125,363 frames
Average number of messages per frame	60.03 messages
Total CPU Utilization	120 CPU seconds
Average CPU Utilization per message	1.8 microseconds



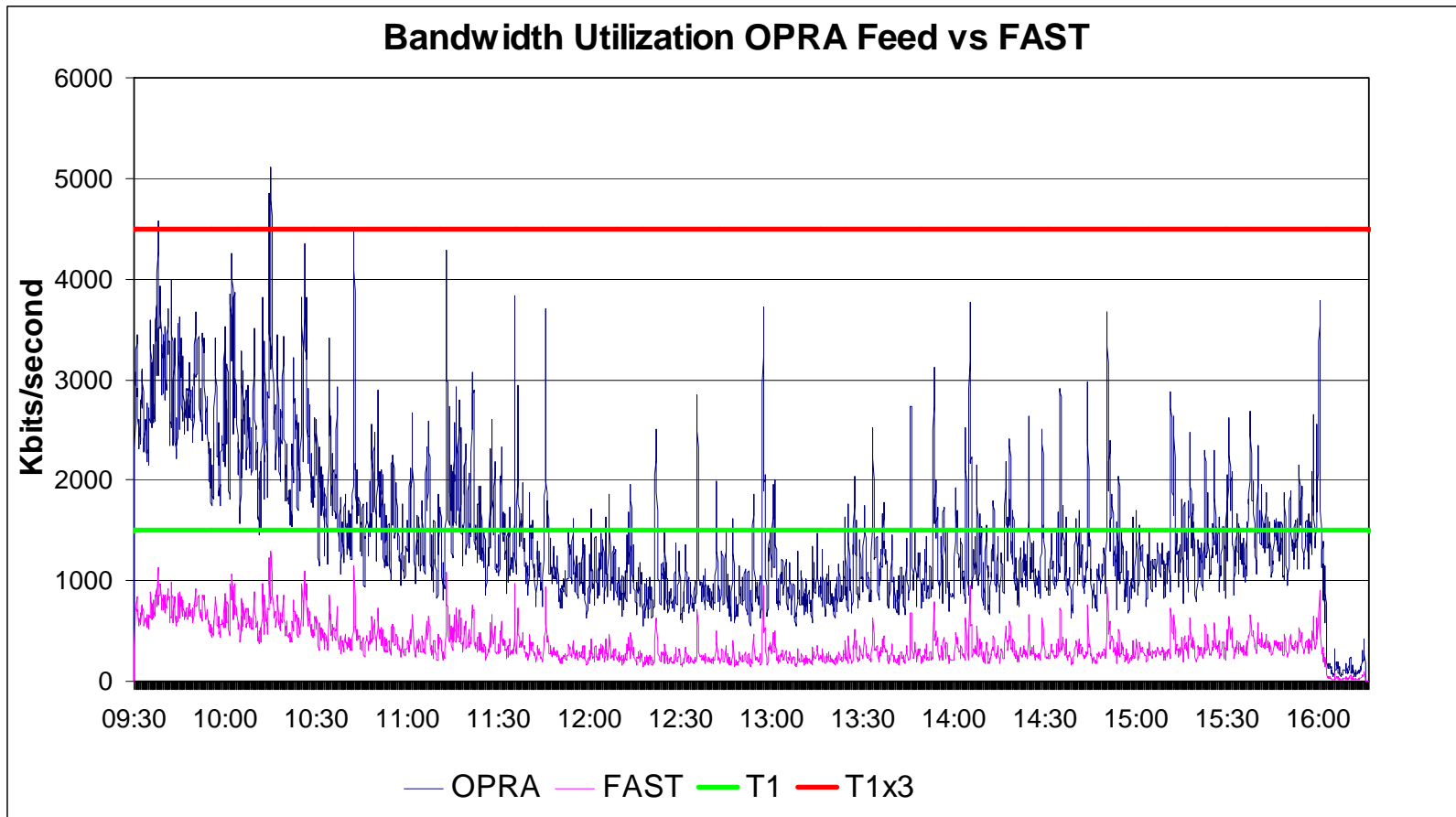
## OPRA Native FAST Performance FAST Protocol - OPRA Compression Ratio Performance





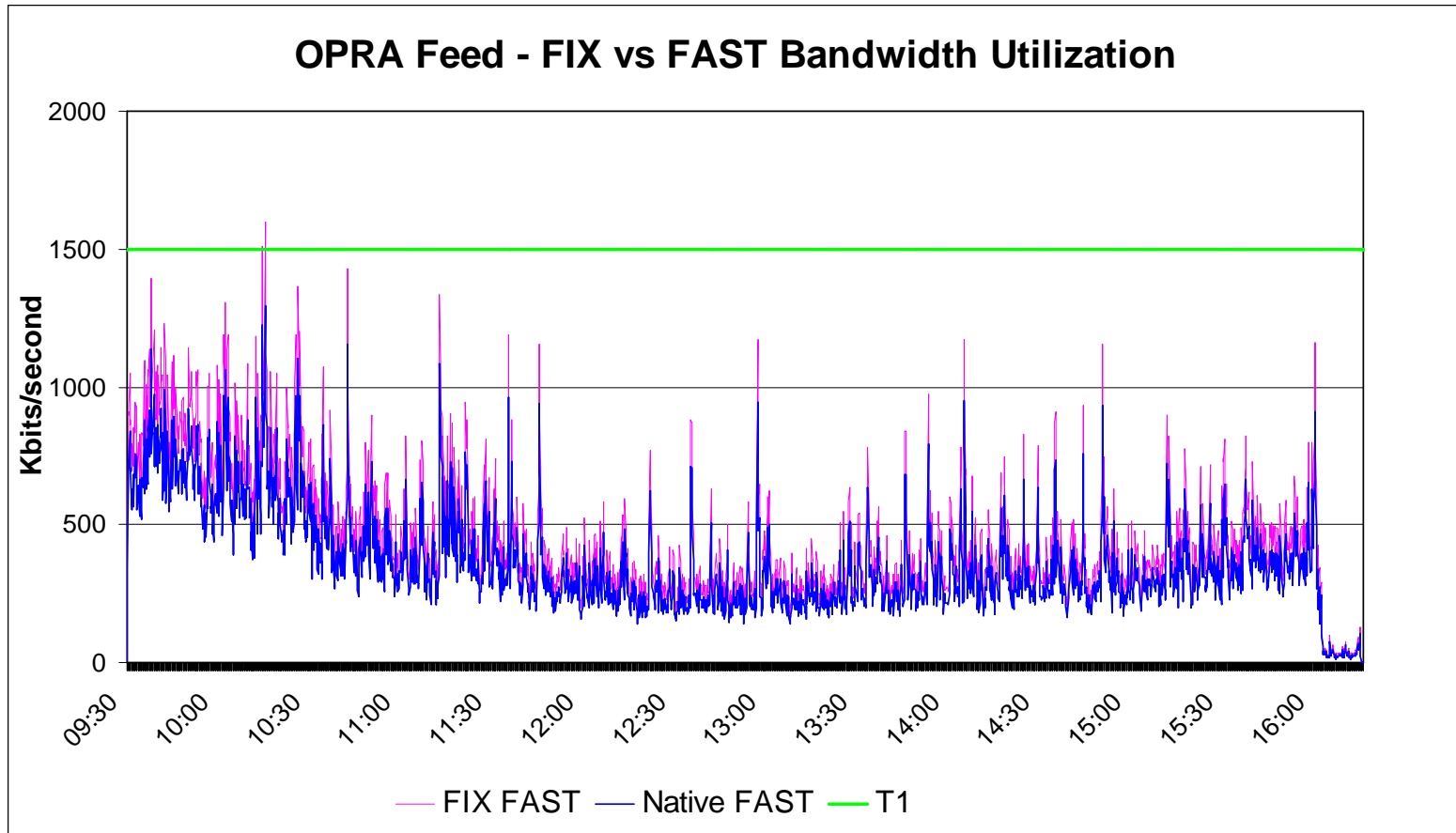
## OPRA Native FAST Performance

### FAST Protocol vs. Native - OPRA Bandwidth Utilization





## OPRA FIX FAST Performance FIX FAST vs. Native FAST – OPRA Bandwidth Utilization



# POC Summary Results

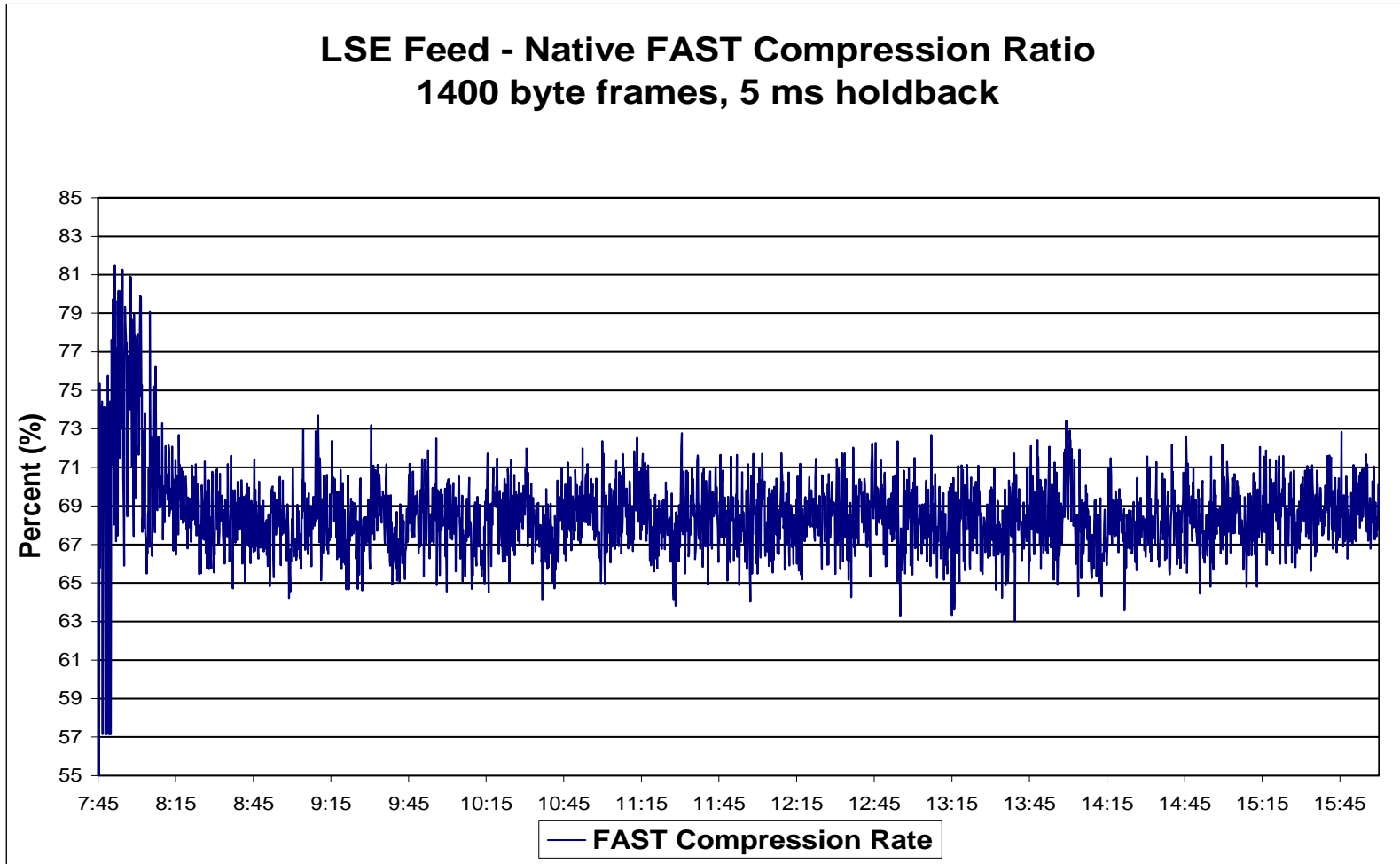


## LSE Native FAST Performance – SETS and SETSmm

Test Category	Result
Trade Date	June 15, 20005
Maximum Compaction Factor (Percent/Ratio)	81.47% / 5.40
Average Compaction Factor (Percent/Ratio)	65.48% / 2.90
Minimum Compaction Factor (Percent/Ratio)	52.94% / 2.12
Frame Size	1400 bytes
Total number of messages	796,110
Average message size before compression	157.7 bytes
Average message size after compression	54.43 bytes
Total number of bytes before compression	125,535,540 bytes
Total number of bytes after compression	43,332,311 bytes
Total number of frames per test	571,811
Average number of messages per frame	1.41
Total CPU Utilization	6.7 CPU seconds
Average CPU Utilization per message	8.38 microseconds

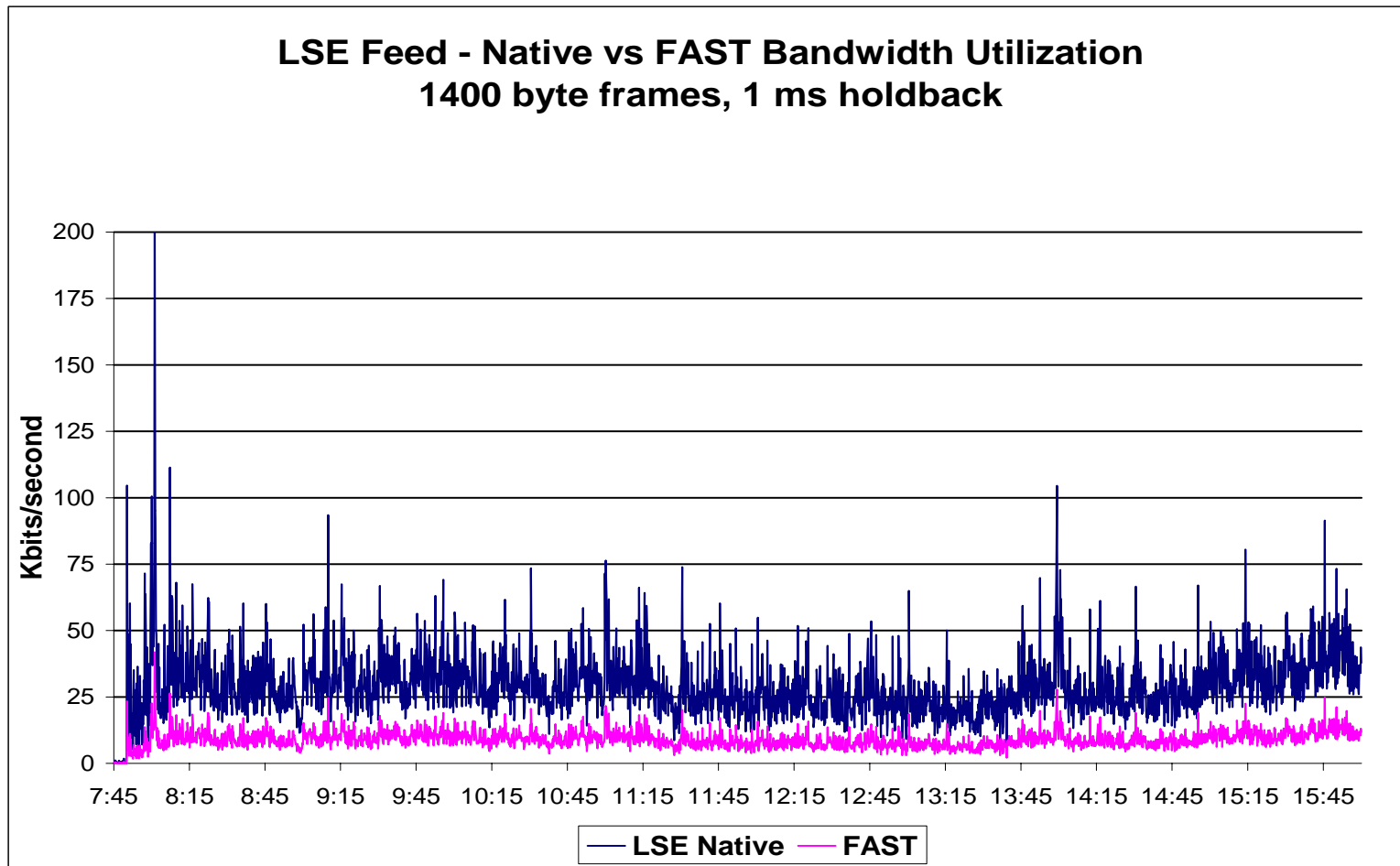


## LSE FAST Performance FAST Protocol - LSE Native Compression Ratio Performance





## LSE FIX FAST Performance FAST Protocol vs. FIX - LSE Bandwidth Utilization





## FAST Protocol – A Basic Example

### Classic FIX Format

```
262=123456 | 55=SP | 48=CME000150112 | 22=4 | 460=7 | 167=FUT |  
200=200312 | 201=1 | 202=1055 | 206=L | 207=2 | 107=SPZ3C1850 | 268=4 |  
290=1 | 269=0 | 270=1050 | 271=2 | 273=104444651 | 272=20030930 |  
290=2 | 269=0 | 270=1050 | 271=1 | 273=104458568 | 272=20030930 |  
290=3 | 269=0 | 270=1030 | 271=4 | 273=104434395 | 272=20030930 |  
290=4 | 269=0 | 270=1020 | 271=3 | 273=104417468 | 272=20030930 |
```



## FAST Protocol – A Basic Example

### Field Encoding Operators

Entry	Description
$\text{f!10}$	Default value 10 if not specified
$\text{f=}$	Copy previous value if not specified
$\text{f+}$	Increment previous value if not specified
$\text{f-}$	Delta value from previous value



## FAST Protocol – A Basic Example

### Implicitly Tagged FIX

#### Message Template

```
262 | 55 | 48 | 22 | 460 | 167 | 200 | 201 | 202 | 206 | 207 | 107  
<290 | 269 | 270 | 271 | 273 | 272>
```

#### Implicitly Tagged Message

```
123456 | SP | CME000150112 | 4 | 7 | FUT | 200312 | 1 | 1055 | L | 2 | SPZ3C1850  
<1 | 0 | 1050 | 2 | 104444651 | 20030930 }  
2 | 0 | 1050 | 1 | 104458568 | 20030930 }  
3 | 0 | 1030 | 4 | 104434395 | 20030930 }  
4 | 0 | 1020 | 3 | 104417468 | 20030930 >
```



## FAST Protocol – A Basic Example

### Implicitly Tagged with Field Encoding

#### Message Template with Operators

```
262 | 55 | 48 | 22 | 460 | 167 | 200 | 201 | 202 | 206 | 207 | 107  
<290+1 | 269= | 270- | 271! | 273- | 272=>
```

#### Field Encoded Message

```
123456 | SP | CME000150112 | 4 | 7 | FUT | 200312 | 1 | 1055 | L | 2 | SPZ3C1850  
< | 0 | 1050 | 2 | 104444651 | 20030930 }  
| | 1 | 13917 | }  
| | -20 | 4 | -24173 | }  
| | -10 | 3 | -16927 | >
```



## FAST Protocol Serialization

### ASCII Representation of 10 Market Data Messages

#### Message Content (separated with |)

A|B|2|3501|300|AAPL|878|14400|2|P|P|ARCAX|21|

45 bytes

A|S|10|25|32000|AOLA|878|14400|2|P|P|ARCAX|22|

46 bytes

A|S|7|3|16000|AOLA|878|14400|1|P|P|ARCAX|23|

44 bytes

A|S|8|38|32000|AOLA|878|14400|2|P|P|ARCAX|24|

...

A|S|3|693|1000|DVID|878|14400|2|P|P|ARCAX|25|

A|S|4|268|4500|EGHT|878|14400|2|P|P|ARCAX|26|

A|S|9|296|197|NVDA|878|14400|1|P|P|ARCAX|27|

A|S|1|8|5000|PCNTF|878|14400|0|P|P|ARCAX|28|

A|S|5|1146|910|VEXP|878|14400|2|P|P|ARCAX|29|

A|S|6|546|1600|ZICA|878|14400|2|P|P|ARCAX|30|



## FAST Protocol Serialization

### Binary Continuation Bit Encoding of the first two messages

HEX	Content
b7 7f	[map]
00	0
02	2
9b 2d	3501
82 2c	300
c1 c1 d0 4c	'AAPL'
06 6e	878
f0 40	14400
02	2
50	'P'
50	'P'
c1 d2 c3 c1 58	'ARCAX'
15	21

**25 bytes**

HEX	Content
37	[map]
01	1
08	8
19	25
81 fa 00	32000
c1 cf cc 41	'AOLA'

**11 bytes**