FACULTY OF TECHNOLOGY

INFORMATION TECHNOLOGY

Telecommunications

FINAL STUDY

Determining Bandwidth for Server in Saxess and Genium Trading Platforms

Name: Birger Nyman
Supervisor: Antti Koivumäki
Instructor: Ari Peltoniemi

Approved: __. __. 2008

Antti Koivumäki
Senior Lecturer

PREFACE

This study was done as a graduation project for Metropolia. As a final study it was challenging, but with the help of the Swedish experts I was able to complete it. I faced many problems and I looked for an answer to the problems sometimes for a long time. But in the end I was able to find solutions to all the problems.

I have learned a lot from trading and FIX technology. I have had a chance to work with the most intelligent people within the exchange technology.

I would like to thank Nasdaq OMX for giving me this final study subject. I would also like to thank my instructor Ari Peltoniemi for guiding me through the project. Also thanks to Tomas Lundqvist and Per-Anders Sahlin. I would like to extend my gratitude to Antti Koivumäki for his help and support and Jonita Martelius for revising the language.

Helsinki, October 7th, 2008

Birger Nyman

INSINÖÖRITYÖN TIIVISTELMÄ

Tekijä: Birger Nyman

Työn nimi: Kaistanleveyden tarpeen arviointi yhdelle serverille Saxess ja Genium kaupankäyntijärjestelmissä

Päivämäärä: 7.10.2008	Sivumäärä: 39 s. + 3 liitettä
Koulutusohjelma: Tietotekniikka	Suuntautumisvaihtoehto: Tietoliikennetekniikka

Työn valvoja: Antti Koivumäki, Yliopettaja

Työn ohjaaja: Ari Peltoniemi, Application Manager

Tässä insinöörityössä arvioidaan kaistanleveyden suuruutta yhtä kaupankäyntiserveriä kohti nykyisessä Saxess järjestelmässä ja tulevassa Genium järjestelmässä. Saxessissa voi käydä vain niin sanottua käteiskauppaa, muihin arvopaperimarkkinoihin on eri ohjelmat. Geniumissa taasen voi käydä käteiskauppaa ja muuta arvopaperikauppaa. Työn tulosten vertailukelpoisuuden vuoksi Geniumista arvioidaan vain käteiskaupan osuus. Päämääränä on arvioida kuinka paljon kaistaa tarvitaan tietyllä osakekauppamäärällä yhtä välittäjän serveriä kohti.

Työ perustuu aikaisempiin mittauksiin osakekaupankäyntiverkosta ja on täysin teoreettinen. Työssä esitellään mitä on osakekaupankäynti ja mitä viestejä lähetetään milloinkin. Näistä viesteistä muodostuu tietty määrä liikennettä tietoverkossa josta arvioidaan keskimääräinen kuorma. On huomattava, että lopputulos on karkea arvio, koska yhtä myynnissä olevaa osaketta kohden saattaa tulla yksi tai esimerkiksi kahdeksan ostotarjousta ennen kuin lähetetään viesti, että kauppa on muodostunut.

Saxess on nykyinen kaupankäyntijärjestelmä, eikä sen teknologiaan paneuduta sen tarkemmin. Genium on uusi korvaava järjestelmä, ja työssä esitellään teknologiset yhteydenpitoperiaatteet joiden pohjalle Genium on rakennettu.

Avainsanat: : Saxess, Genium, osake, kaistanleveydenmäärittely

Metropolia

ABSTRACT

Name: Birger Nyman

Title: Determining Bandwidth for Server in Saxess and Genium Trading Platforms

Date: October 7th, 2008 | Number of pages: 39 p. + 3 in appendix

Department: Information Technology | Study Programme: Telecommunications

Instructor: Antti Koivumäki, Senior Lecturer

Supervisor: Ari Peltoniemi, Application Manager

In this final study an approximation of how much bandwidth one server requires in Saxess and Genium environment was studied. In Saxess it is possible to trade only in so called cash markets, for other financial instruments there are other programs. In Genium it is possible to trade in cash markets and in all the other markets, for example derivates. For the final results to be comparable only cash market traffic was analyzed in Genium. The goal was to determine how much bandwidth was needed for one member server when a certain number of trades were executed.

The study is mostly based on previous studies about stock exchange network and is completely theoretical. Trading and messages sent are explained. These messages generate a certain amount of network traffic and from this traffic a rough bandwidth load approximation was calculated. It must be pointed out that the final result of this study is a rough approximation due to the nature of trading. For each sell order there can be one or for example eight buy orders until a trade execution message is sent.

Saxess is the present trading platform and its technology is not studied here. Genium is the next platform and in this study the basic communication technologies behind it are explained.

Keywords: Saxess, Genium, stock, bandwidth, trading

TABLE OF CONTENTS

Abbreviations

FAST FIX Adapted for Streaming

FIX Financial Information Exchange

LSB Least Significant Byte

MBL Market By Level

MBO Market By Order

MSB Most Significant Byte

OSI Open System Interconnection

PMAP Presence Map

TCP/IP Transmission Control Protocol/Internet Protocol

TE Trading Engine

WAN Wide Area Network

XMP Exchange Message Protocol

XTP Exchange Transaction Protocol

1 INTRODUCTION

The purpose of this final study was to determine how much bandwidth is needed for one trading bank server when a certain number of transactions take place. Today Nasdaq OMX uses the Saxess platform for trading and it is changing to Genium platform. It is known how much bandwidth is needed in Saxess but one has to estimate how much traffic the same number of trades generate in Genium. In order to do this, message type analysis was done to find out which messages create the biggest bandwidth load. The concept of trading was introduced to show how much one trade generates traffic. When a trading bank enters a new sell or buy order it is disseminated to the other members. These orders and executed trades are analyzed to determine bandwidth load.

The study was done while the author was working for Nasdaq OMX. Most of the knowledge to do this study was got from the colleagues by email interviews. Some internal websites were very good sources for information about Saxess. Nasdaq OMX has done one very good study about adding FAST compression technique to the existing Saxess trading platform and this study turn out to be maybe one of the most important study, which was closely related to this final study. In this final study Financial Information Exchange FIX and FIX Adapted for Streaming FAST are studied. There is a website where is distributed material about FIX and FAST, www.fixprotocol.org. The material on the website is updated frequently and one can follow the recent development of the FIX technology. This material from colleagues, internal websites and external websites were studied and analyzed.

In this study the technology of the Saxess trading platform is not studied. Saxess is the old system and it will be replaced by Genium and therefore it was left out. Instead, how the compression works in Genium between the Central Server and the Client Server is explained in detail.

In chapter two is described what Saxess is and in what way the new Genium will resolve some problems that are encountered frequently in Saxess. Chapter three covers the details as to what is a trading network and how the market data feed is generated. Some trading examples are given to illustrate in detail from which messages the market data feed is consists of.

In chapter four the Saxess bandwidth calculations are presented. Chapter five covers the Genium subject and it is divided into seven sections. In these chapters the details of Field Encoding, Transfer Encoding, PMAP and Stop Bit are covered, they are the key elements of the FIX technology. Also the Genium bandwidth calculations for market data feed are presented in chapter 5. In the final part of Chapter 5 there is a figure where the bandwidth consumption of the Saxess and Genium are presented. The sixth chapter discusses the results of this study.

2 SAXESS

In this chapter it is briefly explained in which countries Nasdaq OMX operates. A very simplified figure is shown about the similarities and differences in Saxess and Genium just for the reader to understand where the research phenomenon takes place. Some reasons why the Saxess will be replaced by Genium are explained.

The name of Nasdaq OMX's trading program is Saxess. In Saxess it is possible to trade stocks. There is a central trading server running Saxess Server version, desktops are using Saxess Client version to be in contact to the central server. The following exchanges are part of this Nordic exchange, Copenhagen, Stockholm, Helsinki, Reykjavik, Tallin, Riga and Vilnus. The Nordic countries create one market area and Baltic countries a one of their own. XTP, eXchange Transaction Protocol is the connection technology behind Saxess. XTP uses LZ compression method to reduce data sent between Central Saxess Server and Client Servers. Genium is the new trading platform developed by OMX and it will replace Saxess. In Genium the client connection will be based on FIX technology and it will use FAST, Fix Adapted for STreaming to reduce data sent between the client server and central server. Figure 1 illustrates the connection between the Central Server and the Client Server in both Saxess and Genium.

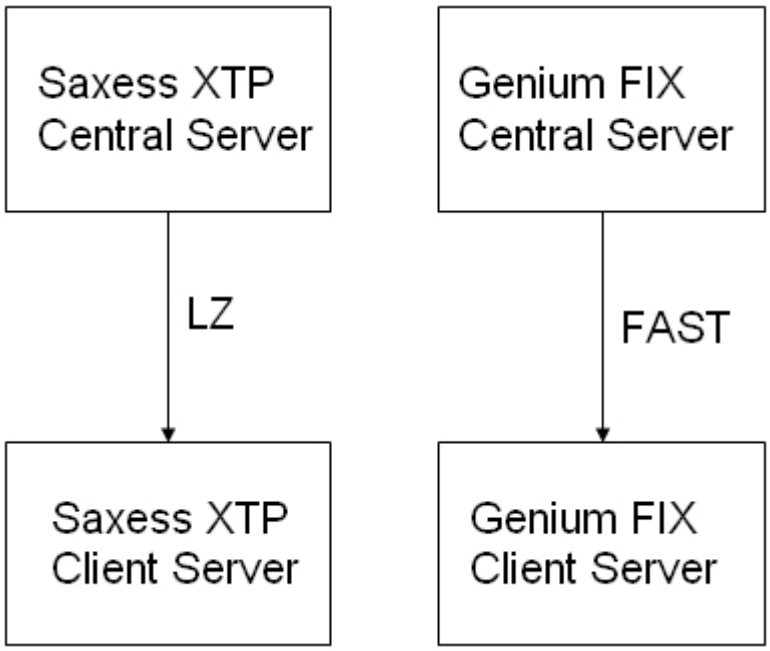

Figure 1. Basic consept in Saxess and Genium

As can be seen in Figure 1, in both, Saxess and Genium, there is a Central Server and a Client Server. Both, Saxess and Genium are programs which are installed to servers; there are different versions for the Central Server and the Client Server. LZ is the compression method used in Saxess between the Central Server and the Client Server. In Genium FAST compression method is used to reduce data sent between the Central Server and the Client Server. The connection between the Central Server and the Client Server is the one which is researched in this study. In Saxess TCP/IP is used to establish and to maintain the connection and LZ is only the compression method. At the moment the Saxess network is highly loaded and it is on its limit to handle all the traffic. When ever there is a lost IP-packet it must be resend. This creates a problem, if the network is already full of traffic this in its self increases the probability to lost IP-packets. Resending the lost IP-packets loads the network even more and then there are even more lost IP-packets and more resending. The network can end up to be congested. One can increase the bandwidth but in some point it is not feasible anymore, especially what comes to costs to maintain such a big bandwidths. Genium was developed to solve this issue, its compression is based on FAST method and it has better compression ratio than LZ used in Saxess.

3 TRADING NETWORK

This chapter is divided into body and two sub sections. In the body part trading network is explained in more detail to show how servers are located in respect to each other. In the first sub section is explained what is market data, how it is generated and how market data is disseminated to different participants in the network. In the second sub chapter is explained the details of the most common messages in the Saxess network. One should understand which are the four most common messages and what each message does and when each of the four message is generated and why. This is illustrated by doing some trades and explaining each step in the trading process.

In Figure 2 is presented the trading network setup in more detail to illustrate how the Trading Engine, which could be also called the Central Server, is located in respect to Client Trading Server, for example.

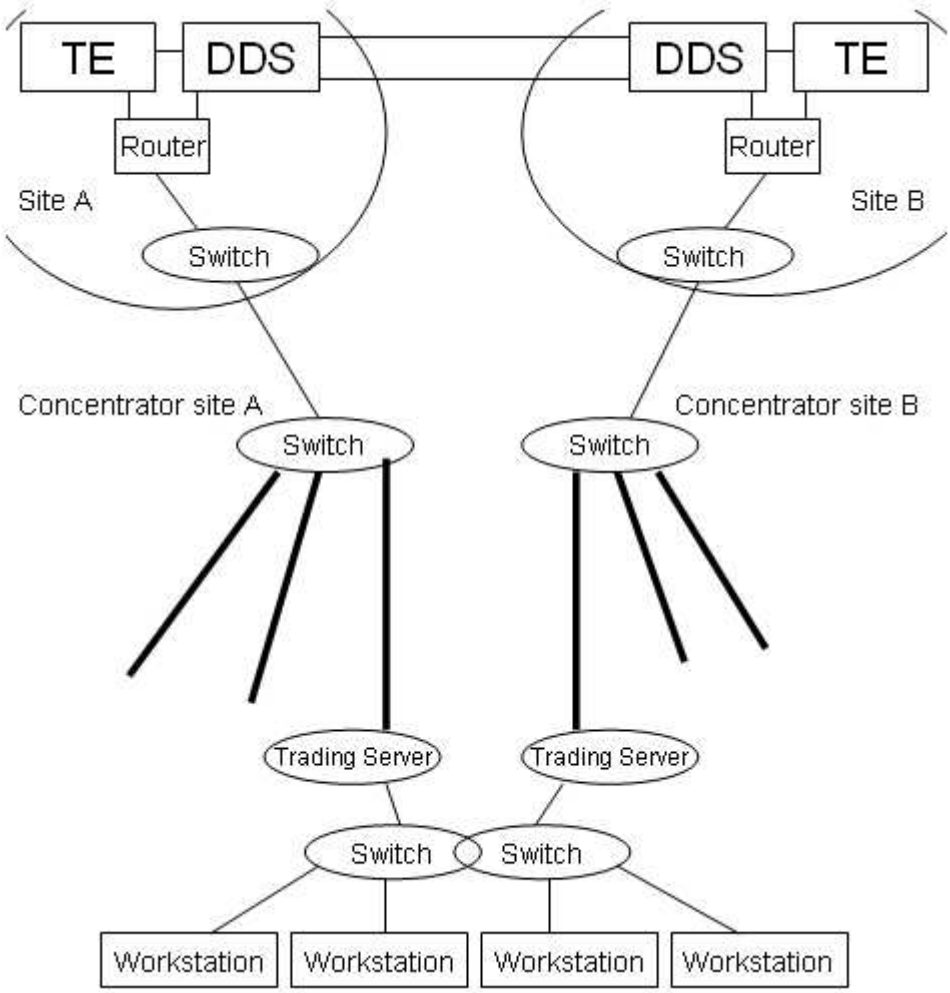

Figure 2. Trading network setup

To achieve redundancy the whole trading system is duplicated from workstations onwards. The workstations in the Figure 2 represent workstations in one member premises [2], p.8. Workstations are connected to a switch and the switch is connected to the Trading Server in member premises and the Trading Server is connected to a switch on Concentrator site A. All the other members are connected to the switch on Concentrator site A in the same way. Concentrator site B is on stand by if there is a failure on site A. Each inserted order and executed trade generates messages which are disseminated to the other members. There is a configurable limit of how much the Trading Engine can push out data to disseminate it to the members. If this limit is reached the Trading Engine will not take any new orders in. During time more members are connected to the network and the output rate must be recalculated and increased. In this study the bandwidth requirements for one member server when using Saxess and Genium are compared.

3.1 Concept of Trading

This sub chapter gives an overall picture why and when trading messages are disseminated and how the dissemination process goes. What is the market data feed and how it is generated is explained. In Figure 3 is illustrated a trading network in order to show how market data feed is generated.

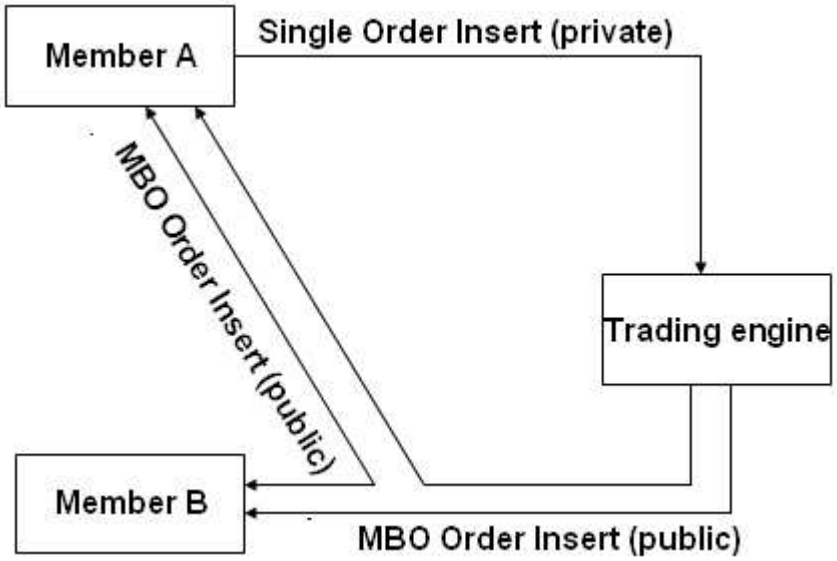

Figure 3. Simplified trading network

In Figure 3 there is a simplified trading network showing how market data is generated and how it is disseminated to the members in the trading network. There are two members A and B, both sending sell and buy orders to the Trading Engine. From the Trading Engine these orders are disseminated to the other members in the network. This is explained in more detail in the next paragraph.

When one wants to buy stocks from the stock markets one has to go to a bank to do this. No individual person can be directly connected to the trading network as so called members a.k.a banks can be. Now is given one example how the trading takes place. The Member A will insert an order into the trading system. Person A wants to buy 1000 Nokia stocks for a price of 15€ each. The bank will insert this buy order into the trading network. The buy order travels in a so called private part of the data cable to the Trading Engine where the matching process between the buy and sell orders takes place. The orders coming from the Member A are not visible to the other

members until they have gone through the Trading Engine. In the Trading Engine person A's order is placed into an order book. All the other sell and buy orders are inserted into this same order book. This order book is refreshed frequently and the information is disseminated to all the other members. Now Member B has received a sell order from their customer person B and the sell order is inserted into the order book in the same way as customer A's is. Customer B wants to sell 1000 Nokia stocks for a price of 16€ each and customer A wants to buy 1000 Nokia stocks for a price of 15€ each. Trade will not be executed due to price difference. Member B is authorized to decrease Customer B's sell offer down to 15€ per stock and in this scenario Member B does that to get Customer B's stocks sold. Member B inserts a15€ sell order and a match is made and a Trade Execution message is sent out to all the members. All the other members know now this trade is executed. Members are sending these sell and buy orders to the Trading Engine and these orders create order books and this information is sent as a public market data feed to the other members.

3.2 Trading Messages

In this sub chapter the four most common messages in the Saxess trading platform are explained. This is done by giving some trading examples. These four messages create most of the bandwidth load. First is illustrated an order book where there are already some sell and buy orders. Then into this order book some new sell and buy orders are inserted and matches are made. By using several tables each step is explained, how the trading process goes and when each of four messages is generated. Table 1 illustrates an order book and into this order book is entered new sell and buy orders, these steps are illustrated in Tables 2 to 9.

Table 1. Order book

NOKIA			
Buy units	Buy price	Sell price	Sell units
1000	15	16	500
2000	15	16	1000
500	15	17	1000
100	14	18	600

In Table 1, there are four buy orders and four sell orders in an order book. In this order book sell and buy orders are on such price levels that trade will not occur. Each order in this order book has generated one MC_Order_Insert message which is disseminated to the other members in the public market data feed. [1], 6.3.1.2. In Table 2 is entered a new buy order into the order book.

Table 2. Entering a new buy order

NOKIA			
Buy units	Buy price	Sell price	Sell units
500	16	16	500
1000	15	16	1000
2000	15	17	1000
500	15	18	600
100	14		

In Table 2, member X has received the public market data about this order book, which was shown in Table 1, and decides to enter an order to buy 500 stocks for a price of 16€ each. This buy orders is placed into the order book and a match is made. In Table 3 the buy and sell order is removed from the order book.

Table 3. Order removed from the order book

NOKIA			
Buy units	Buy price	Sell price	Sell units
1000	15	16	1000
2000	15	17	1000
500	15	18	600
100	14		

In Table 3 the matched buy and sell orders are removed from the order book and a Trade message is disseminated to the other members in the public market data feed. Here a full match order took place and it generated two messages, MC_Order_Cancel and MC_Trade. MC_Order_Cancel message is sent out always when an order is removed from the order book and MC_Trade message is sent out always when a trade is made public.[1], 6.3.1.4 & 6.5.1. Orders can be fully matched as happened in the example in

Table 3. It can be so that order is only partly matched. In Table 4 a partly matched order takes place.

Table 4. Partly matched order

NOKIA			
Buy units	Buy price	Sell price	Sell units
1000	15	16	500
2000	15	16	1000
500	15	17	1000
100	14	18	600

In Table 4 there is the same start situation in the order book as in the fully matched order example which was shown in Table 1. In Table 5 is entered such a big buy order that it must be matched with two sell orders.

Table 5. Entering a big buy order

NOKIA			
Buy units	Buy price	Sell price	Sell units
1200	16		
1000	15	16	500
2000	15	16	1000
500	15	17	1000
100	14	18	600

In Table 5 member X enters an order to buy 1200 stocks for a price of 16€ each. When there are several sell orders in the order book for the same price and same stock the sell order first placed to the order book will be first executed and in this case it is the sell order of 500 stocks for 16€. Because the buy order was 1200 stocks, 700 stocks more are bought from the sell order of 1000 stocks for 16€ thus leaving 300 stocks to the order book for further sell and this is a partly matched order when only part of the sell order is bought. In Table 6 is illustrated the order book after partly matched order.

Table 6. Order book after partly matched trade

NOKIA			
Buy units	Buy price	Sell price	Sell units
1000	15		
2000	15	16	300
500	15	17	1000
100	14	18	600

In Table 6 there is the order book after one fully matched and one partly matched order. The partly matched order generates one more message compared to the fully matched order, MC_Order_Insert. MC_Order_Cancel message is sent out always when an order is removed from the order book and MC_Trade message is sent out always when a trade is made public and when an order is partly matched Trading Engine enters automatically a new order into the order book for the remaining 300 stocks using message MC_Order_Insert. This detail must be remembered when calculating band-widths for the market data feed. A member can update its own order and this is illustrated in Table 7 and Table 8.

Table 7. Order book update

NOKIA			
Buy units	Buy price	Sell price	Sell units
1000	15	16	500
2000	15	16	1000
500	15	17	1000
100	14	18	600

MBO_Order_Update message is disseminated to all the members when one member updates individual order in an order book. In Table 7, there is a buy order of 100 units and its buy price will be reduced from 14€ down to 13€. [1], 6.3.2.3. Table 8 shows the order book after the update is done.

Table 8. Order book after update

NOKIA			
Buy units	Buy price	Sell price	Sell units
1000	15	16	500
2000	15	16	1000
500	15	17	1000
100	13	18	600

In Table 8 there is the order book after the update was done. Another way to disseminate market data is Market By Level, where all orders on the same price level are summed together as if it was one order. This is illustrated in Table 9.

Table 9. Orders on the same price level

NOKIA			
Buy units	Buy price	Sell price	Sell units
3500	15	16	1500
100	14	17	1000
		18	600

In Table 9 buy orders on the same price level are summed together. In Table 7 buy orders of 1000, 2000 and 500 are all on the price level of 15€ therefore those are summed together creating a buy order of 3500 which is shown in Table 9. In Table 7 there is only one buy order on the price level of 14€ therefore in Table 9 there is only 100 stocks on the price level of 14€. The same is done for the stocks which are for sell. From Table 7, 500+1000=1500 stocks are on the price level of 16€, 1000 on the price level of 17€ and 600 on the price level of 18€ as is shown in Table 9. In the study of Tomas Lundqvist and Rolf Andersson MBO data in the feed takes 88,5% of the total bandwidth and MBL 3,9%. Market By Level cannot be used for trading because individual sell and buy orders cannot be identified from the feed. In this study the focus is only in MBO. In the previous examples the message names are from the Saxess system which is the existing trading platform and these message names are not applicable in the Genium trading platform but these messages and their names clearly bring out the principle of trading which must followed in any trading platform.

In the following chapter an estimate of how much bandwidth is needed for one member server is presented.

4 SAXESS BANDWIDTH CALCULATIONS

In this chapter the calculations of the bandwidth requirement for one member server are presented. Previous studies from the Saxess trading network are used to do this. It is a very straightforward process to calculate the bandwidth. First is calculated the application traffic and after this the frames are added. To calculate the application traffic the average load of the four most common messages has to be determined. Also in this chapter is illustrated how the Saxess messages are located in the OSI reference model. At the end of this chapter is Figure 5 which illustrates bandwidth/trades ratio from today's requirements up to over double what is possible to execute today. Table 10 shows the four most common messages in the Saxess platform and the total size of each message type.

Table 10. Trading messages

Message type	Number of messages	Message distribution	Total size in bytes	Size distribution
MBO OrderInsert	325 365	41,2%	41 321 355	44,4%
MBO OrderCancel	233 962	29,7%	17 079 226	18,4%
Trade	98 412	12,5%	16 927 224	18,2%
MBO OrderUpdate	48 081	6,1%	7 019 826	7,5%

In the study of Tomas Lundqvist and Rolf Andersson the effect of adding FAST compression technique to the Saxess system without first converting the messages into FIX format was studied [3]. Table 10 is based on this study, and from this table can be seen the amount of messages and trades during one day. These four message types MBO_Order_Insert, MBO_Order_Cancel, Trade and MBO_Order_Update create 88.5% of the disseminated market data feed. From these figures is calculated an average of how much traffic is generated in respect to one executed trade. After this traffic analysis, Figure 5, showing Trades/Bandwidth ratio can be presented. In Table 11 is presented how many messages there are for one trade in each message type.

Table 11. Message types per one trade

MBO_Order_Insert	**325 365 / 98 412**	**3.30615 orders per one trade**
MBO_Order_Cancel	**233 962 / 98 412**	**2.37737 cancels per trade**
MBO_Order_Update	**48 081 / 98 412**	**0.48856 updates per trade**

Table 11 presents how many MBO_Order_Insert, MBO_Order_Cancel and MBO_Order_Update messages are sent in respect to one executed trade. Where as Table 12 shows the message size in bytes.

Table 12. Size per message type

41 321 355 / 325 365	**127 bytes for one MBO_Order_Insert**
17 079 226 / 233 962	**73 bytes for one MBO_Order_Cancel**
16 927 224 / 98 412	**172 bytes for one Trade**
7 019 826 / 48 081	**146 bytes for one MBO_Order_Update**

The size of one message in all four message types is shown in Table 12. In Table 13 the outcome of tables 11 and 12 is multiplied.

Table 13. Message size per trade

127 bytes * 3.306615	**419.88105 bytes MBO_Order_Insert**
73 bytes * 2.37737	**173.54801 bytes MBO_Order_Cancel**
172 bytes * 1	**172 bytes Trade**
146 bytes * 0.48856	**71.32976 bytes MBO_Order_Update**

Table 13 summarizes how many messages there are per one trade and the size of each message type. Having this information, it is known how much there is application traffic per on trade. Then one has to reduce the application data by compression level and add XMP and TCP/IP frames to get the

real bandwidth per one member server. Table 14 summarizes the tables 11, 12 and 13.

Table 14. Application messages size per one trade

420 + 174 + 172 + 72 = 838 bytes

Table 14 shows the message size in each message type in respect to one executed trade. MBO_Order_Insert message creates 420 bytes, MBO_Order_Cancel message creates 174 bytes, Trade message creates 172 bytes and MBO_Order_Update creates 72 bytes in respect to one trade. Altogether 838 bytes is generated in average when one trade is executed. LZ-compression is capable of reducing the size of emitted data by 73%, therefore 838 bytes is reduced down to 838*0.27 = 226 bytes. Converting this into bit size makes it 226*8 = 1808 bits. Table 15 shows how the Saxess is located in the OSI model.

Table 15. Saxess in the OSI model

OSI LEVEL	**SAXESS**
7 APPLICATION	**XTP**
6 PRESENTATION	**XMP**
5 SESSION	**XMP**
4 TRANSPORT	**TCP**
3 NETWORK	**IP**
2 DATA LINK	
1 PHYSICAL	

Table 15 shows how the XTP and XMP are located in the OSI model. In Table 15 can be seen that to calculate the size of one Saxess message, XTP message on the 7th layer has to be added into XMP frame which is providing services on the layers 6 and 5 and this XMP frame is added into TCP/IP frame which providing services on the layers 4 and 3. Exchange Message Protocol XMP provides support for compression and encryption. These XMP frames are sent over a TCP/IP session. [4] Figure 4 illustrates how the XTP, XMP and TCP/IP are added together.

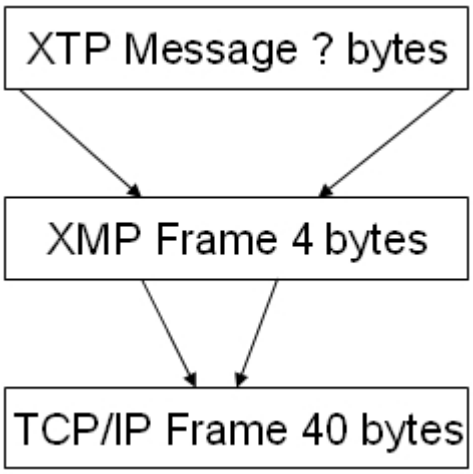

Figure 4. Adding XTP application message into frames

In Figure 4 there is a question mark next to the XTP Message. It is there because it is not known how big the XTP message which is added into XMP frame is. But XMP and TCP/IP frames have always the same size. There are about 6 XTP messages per one trade, each of them encapsulated in XMP frame and this is sent over a TCP/IP connection. XMP frame adds four bytes and TCP/IP 40 bytes into each message. One has to add XMP frame bytes and TCP/IP bytes into each 6 XTP messages and 1 trade message. 7x4+7x40 = 308 bytes which is 308x8 = 2464 connection bits and when adding the average application traffic of one trade 1808 bits, thus one trade generates about 2464+1808 = 4272 bits. For one member server is reserved 450 kbps which makes 450 000 bits / 4272 bits = 105 trades per second. In Table 16 is presented how many trades can be executed with different server bandwidths.

Table 16. Trades per member server when different bandwidth per server

Server bandwidth	Traffic per trade	Trades
450 000 bits	4272 bits	105
675 000 bits	4272 bits	158
900 000 bits	4272 bits	210
1 120 000 bits	4272 bits	262

In Table 16 the different bandwidths per server and how many trades can be executed on each bandwidth are presented. The need to increase output rate comes from the increasing number of incoming sell or buy orders in one second. In Figure 5 is presented the maximum number of trades which can be executed on each bandwidth rate.

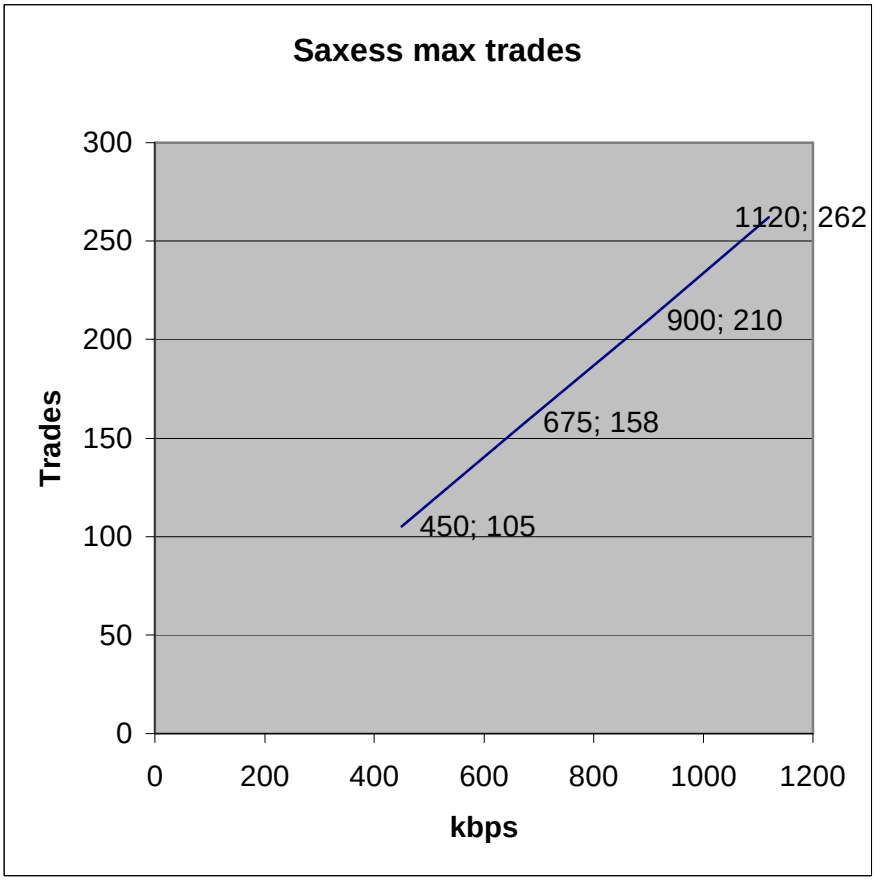

Figure 5. Bandwidth per member server in respect to trades

In Figure 5 the figures next to the curve present the number of trades in respect to kbps, first is the bandwidth in kbps and then the number of trades.

5 GENIUM

This chapter is divided into 7 sections. The first describes what is the FIX. The second chapter gives an introduction to template technology which is used in Genium to reduce data sent between the Central Server and the Client Server. In the third section is given an introduction to PMAP and Stop Bit which are covered in more detail in the later chapters. PMAP and Stop Bit are the most important details in Genium with the template technology. In the fourth section templates are explained in more detail. A Template is a group of data fields which have predetermined data types and operators for each field. Data is inserted into these data fields and then a program such as Genium performs actions based on information it is getting from the template. The fifth section illustrates how the Field Encoding works. To understand Field Encoding one should have an understanding of PMAP and templates which are explained in Sections 5.3 and 5.4. In Section 5.5 one message is sent and each step in the Field Encoding is explained. Field Encoding removes unnecessary data fields and is the core of FAST to create compression. For the remaining data fields Transfer Encoding is performed and this is described in the 6th section. Also in the 6th section the Stop Bit is explained in great detail. PMAP and Stop Bit are parallel phenomena. There is no one without the other. The 7th section covers the Genium bandwidth calculations.

5.1 FIX

In this chapter is described what is FIX and who are behind it. FIX is an acronym for Financial Information eXchange. FIX defines a series of message specifications which are used in electronic trade messaging. FIX has been designed in collaboration with banks and exchanges. The developers share a vision of a global messaging standard for automated trading. The reason why FIX is so successful and widely used is the development efforts of its member firms. Above all is the FPL FIX Protocol Limited and companies can be members of this. Global Steering Committee is the one which is controlling all the subcommittees where the companies co-operate to develop FIX. On a website www.fixprotocol.org one can look for more information. Figure 6 illustrates a basic FIX message.

FIX based network

BeginStr	SeqNum	SenderID	SendingTime	Price	Symbol
8=FIX.4.4 |	34=10000 |	49=CLIENT1 |	52=20060126-13:06:58.100 |	44=3400 |	55=FOO1 |

Figure 6. FIX message

In the message in Figure 6 there are several 8=, 34=, 49=, the number is a tag. Each tag is between | 34=10000 | and this is one field. For the messaging to be standardized each field is tagged. This tag defines what type of data string or int is in the field and the functionality description for what purpose one specific field is meant. These fields can be added as blocks to create working trading messages and developers can make their own FIX application to understand each others. On a website www.fixprotocol.org/FIXimate3.0/ one can enter tag numbers to see more detailed description from any tag. The tags are numbered between 1 and 1139 in FIX 5.0 version. In Figure 7 is performed a search for a one tag.

Figure 7. Tag search on the website

In Figure 7 the tag search is illustrated. One can enter any tag number and find out more detailed description from particular field. In Figure 8 is given a more detailed description of the searched tag.

Tag	Field Name	Data Type	Description
35	MsgType FIXML Abbr: @MsgTyp	String	Defines message type ALWAYS THIRD FIELD IN MESSAGE. (Always unencrypted) Note: A "U" as the first character in the MsgType field (i.e. U, U2, etc) indicates that the message format is privately defined between the sender and receiver. *** Note the use of lower case letters ***

Figure 8. Description of tag 35

After entering a tag number FIXimate shows the corresponding field and its description as illustrated in Figure 8. FIX have different versions therefore some tags cannot be found in some versions. In Figure 9 is illustrated how the Order Book Update message is constructed.

Tag No	Field Name	Value	Bytes	Comment
8	BeginString	FIXT.1.1	8	Constant in all messages (removed by FAST compression)
9	BodyLength	100	3	Just a value put here
35	MsgType	X	1	Market Data Incremental Refresh
1137	ApplVerID	8	1	String, here set to value 8
49	SenderCompID	OMXNE-10	8	
56	TargetCompID	CARNEGIE-3	10	
34	MsgSeqNum	1234567	7	
52	SendingTime	20080131-09:27:54	17	Converted to UTC
1180	ApplID	1111	4	Constant identifier for this flow
1181	ApplSeqNum	111450	7	
1350	ApplLastSeqNum	111449	7	
1021	MDBookType	3	1	Means MBO
268	NoMDEntries	1	1	Number of entries in message
279	MDUpdateAction	0	1	Means New entry
269	MDEntryType	0	1	Means Ask
278	MDEntryID	2067	4	String, ID Calculated from Ask level (1 in this case)
55	Symbol	VOLV B	6	
270	MDEntryPx	120.75	6	
271	MDEntrySize	11600	5	
273	MDEntryTime	09:27:54	8	
346	NumberOfOrders	100	3	
10	CheckSum	288	3	String, three byte checksum field

Figure 9. Order Book Update message definition in Genium

To give a glance of what the FIX messages look like, there is an Order Book Update message used in Genium shown in Figure 9. This is a way to define messages, only the data in the Value field is sent with some restrictions.

5.2 Templates

In this section one message is sent to illustrate how data fields are removed from the message which is the core idea of the FAST compression. In this chapter PMAP is mentioned only as a data field which is added into message. Figure 10 illustrates how the first message is sent.

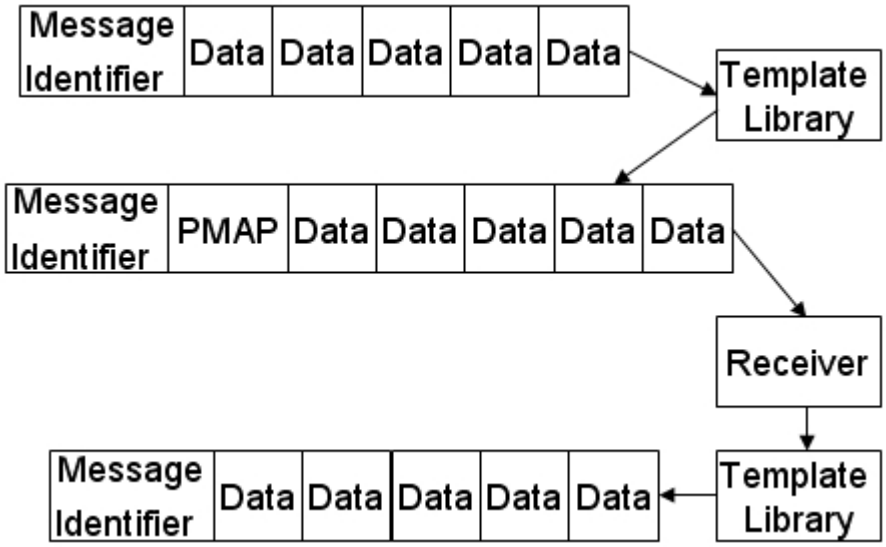

Figure 10. First message sets values to the data fields.

In Figure 10, one trading message is sent from one member to another. The trading program sends the message forward and first it goes into the template library. In the template library the right template is chosen based on message identifier. When the message comes out from the template library PMAP field is added. At the receiver side the message goes again into the template library where the message is reparsed and PMAP field is removed. The first message will set values to data fields to the template on the receiver side. [5] The second message can be shorter as shown in Figure 11.

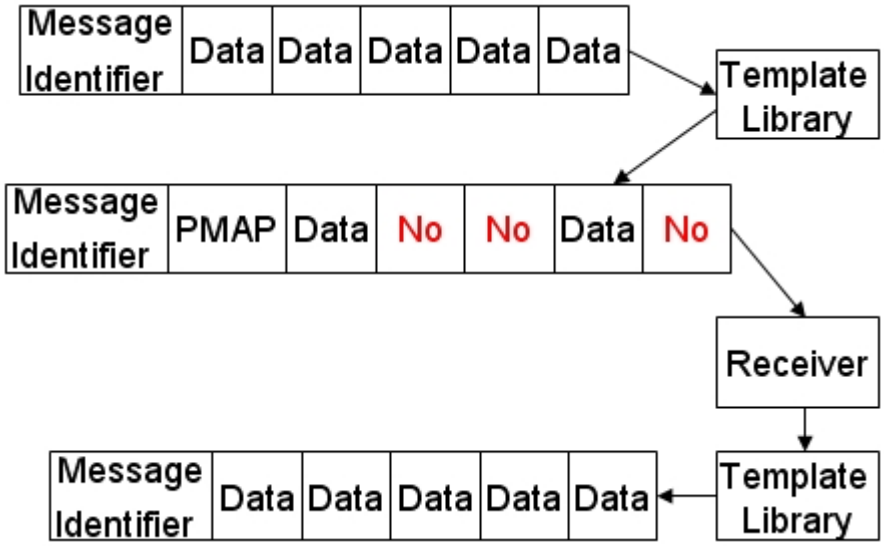

Figure 11. Second message.

In Figure 11, the same member initiates second trading message. When the second message goes into template library it is identified to be same type of message as the previous one. Therefore there is no need to send all the data fields. The program can use the data values from the previous message which were inserted into the data fields of the template on the receiver side. Figure 10 illustrated how five data fields were sent, in Figure 11, only two data fields were sent.

5.3 PMAP and stop bit

In this chapter a short introduction to PMAP and Stop Bit is given. Both of them should be comprehended before reading chapters 5.5 and 5.6. In Figure 12 is illustrated one FIX message.

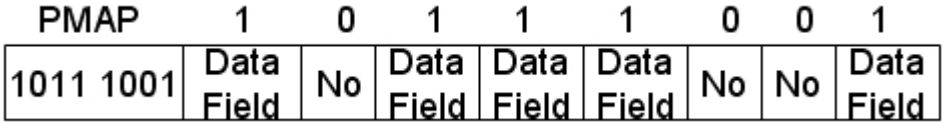

Figure 12. Presence Map

In Figure 12 there is one PMAP field and eight data fields. Into PAMP field is marked if the data field has data inside or not. Presence Map PMAP field identifies which fields are absent and which fields have data inside. 1 is marked into the PMAP field if the data field exists and 0 if the data field is absent in the message. [3], p.2 In Figure 13 is taken one data field from Figure 12 and a deeper look into one data field is taken.

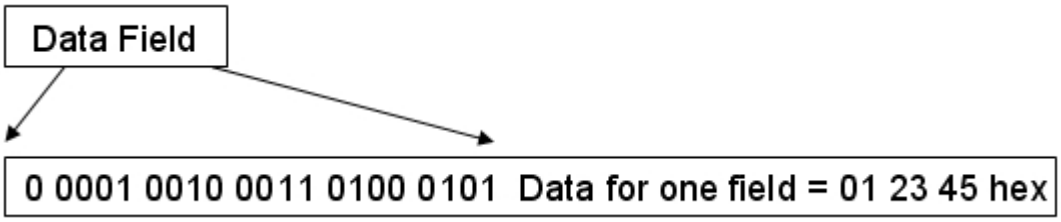

Figure 13. One data field

In Figure 13 is presented the content of one data field in bytes and in hex format. In Figure 14 is presented only the bit part of the Figure 14.

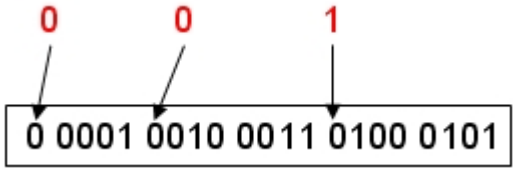

Figure 14. Adding Stop bits

In Figure 14 one stop bit is added into each byte, if the byte is marked by 1 it is the last byte of the field. The stop bit has no data carrying capabilities. In Figure 15 is the data field after the stop bits have been added.

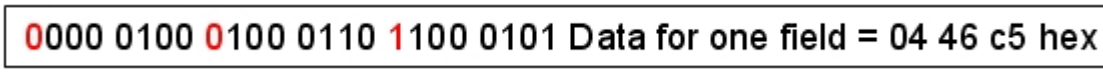

Figure 15. Data field after adding Stop bits

In Figure 15 can be seen that the bit which is replaced by stop bit is just moved left to the next byte. Figure 16 illustrates the same message when it is received.

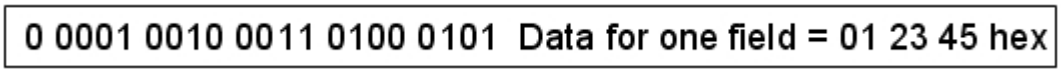

Figure 16. Stop bits are removed on the receiver side

When the message is received the stop bits are removed and the original message is readable as can be seen from Figure 16. [3]

5.4 Templates in Detail

In this chapter the details of templates are explained. A template consists of several data fields and each field is tagged and this tag has a data type and an operator. What data types and operators tags have is explained. FAST uses templates to reduce the size of messages transmitted between sender and receiver. Template defines the structure of each field in a message. Each field is marked with a tag, data type and field operator symbol. In Figure 17 is presented a basic tag.

Figure 17. Tag definition principle

In Figure 17 is shown the tag definition principle, first is the tag number, then comes what data type the data will be what is inserted into the template and then comes the Field Encoding operator. It must be defined what type of data is inserted into a template otherwise the encoder and decoder cannot process the field. The data type symbol is marked after the tag. In Figure 18 are shown the symbols for each data type and their abbreviations.

Symbol	Description
s	String
U	Unsigned integer
u	Unsigned integer supporting a NULL value
I	Signed integer
i	Signed integer supporting a NULL value
f	Scaled number
n	Composite scaled number

Figure 18. Data type abbreviations

In Figure 18 one can read the data type symbols and their description. In Figure 19, all the field operators are presented.

Symbol	Description
!	Default Coding – default value per template
=	Copy Coding – copy prior value
+	Increment Coding – increment prior value
-	Delta Coding – numeric or string differential
@	Constant Value Coding - constant value specified in template
*	Implicit Value Coding – implied field values

Figure 19. Field operators

In Figure 19 the different field operators are presented. Each data field is marked with a field operator, this determines for the encoder and decoder how the field should be coded. For example if a data field is marked by (-) without brackets, the encoder calculates the delta value of two concatenate integer number fields and places this delta value into the next message going to a receiver. The sender and receiver have the same templates and each time a message is sent or received it goes through Field encoding process. Before the message enters the Field Encoding process a right template is chosen for the message. Each message carries a front header containing a template identifier and based on this the identifier right template is taken from the template library to encode the message. In Figure 20, there is a template without any data inside. 999u=1234 is the first field and it defines the ID of the template, 999 is the tag number, u is the data type symbol, = is the operator and 1234 is the number identifying the template. [5]

```
999u=1234|8s!FIX.4.4|9u|35s!X|49s=|34u+1|268u
<279u=|269s=|55s=|167s=|270F-|271F-|346u-|276s=|277s=>
```

Figure 20. Empty template

In the message each field contains a tag, data type symbol, operator and data. This tag identifies what kind of data it is and how to encode it. When a new type of message is sent for the first time from the sender to receiver it is sent almost without Field Encoding, only Constant value fields are removed from the message in the Field Encoding process. The first message is sent almost in its full length in order to get the starting values to the template on the receiver side. There is no need to send Constant value fields because those are same in every following message. The constant value fields have got their values when the templates were distributed statically to both sender and receiver.

In front of each message there is the template identifier and this template identifier is read first when the message arrives to the receiving side and the right template is chosen for decoding and one could see that the Constant value is already placed in the template at the receiving side.

The fields in a message coming from the trading application and in a template are in corresponding order to each others. When a new message is initiated in a trading application such as Genium, the right template is chosen for the message to encode it. The data is placed to the empty data fields in the template and after that the Field Encoding process starts. Every field in a template is predetermined whether it is a Constant, Increment, Copy or Delta and whatever data is inserted into the template data field it is treated by the operator defined in the template. The order must be kept for encoding and decoding to be successful. Each field is processed in the Field Encoding based on the tag value in the data field. The tag value is just a plane number between 1…1139. In the encoder it is defined what to do to a data field containing a particular tag.

5.5 Field Encoding

The Field Encoding is the idea behind the FAST data compression method. Because all the messages are identified by Message Identifiers therefore if the message is identified to be same type as the previous one which was sent, the Genium program knows that previous message has set the data values to data fields and sends only those fields which are different from the previous message. In this chapter is given one example about this. In Figure 21 is described how one message is sent and how the Field Encoding works. First the message goes into Field Encoding where the fields are removed and after that Transfer Encoding is performed. Transfer Encoding is explained in Chapter 5.6.

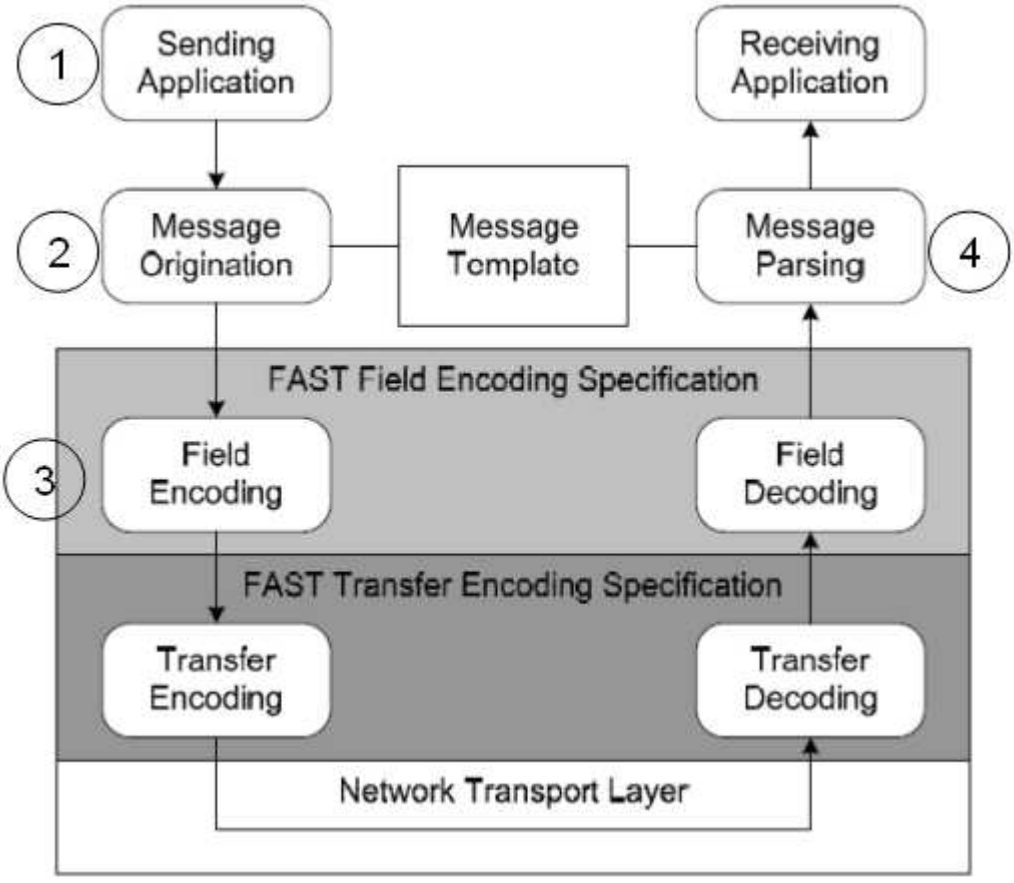

Figure 21. Process of sending message

1. Sending Application generates a trading message

2. In front of this message there is a template identifier. Based on this identifier the right template is chosen. Both sides have the same templates and when the message arrives to the other side and based on message identifier, bytes in the message are placed back to the right template.

3. Each field is marked with a tag number. This tag identifies the data type of the field and how to code it. The order of tags/fields is defined in the template. Each field must be defined by tag number otherwise FAST coder do not know what to do with those bytes. For example if the field is marked by tag 34 it identifies the field to be Incremental. Here, in this example, SeqNum field is defined to be Incremental. When the message is sent for the first time, Sequence Number 10 000 is sent over all the way to the receiver and placed to the SeqNum field in the template at receiver side, this requires 5 bytes. Each message has a message identifier, therefore when the same message is sent again it can be identified to be the same message type at the sending side and when doing the Field Encoding, Sequence Number 10 000 will not be sent therefore the SeqNum field will be empty. PMAP Presence Map field is added to the message at this point. In this field one bit is reserved for each field in the message, 1 when the field has data inside and 0 when the field is empty. Now this message is sent to the receiver but the SeqNum field is totally empty of data.

4. At the receiving side is the same template and it is coded to increment the number by one when a new message is received and the field is tagged as 34. When the message is received for the second time number 10 000 is incremented to 10 001. In this way the second message is 5 bytes smaller because the number 10 001 was never sent in the data cable. [5] In these four steps the Field Encoding was explained. Step by step it was explained how fields can be removed without any data loses, even the data field was never sent in the data cable. Figure 22 shows how several fields are removed.

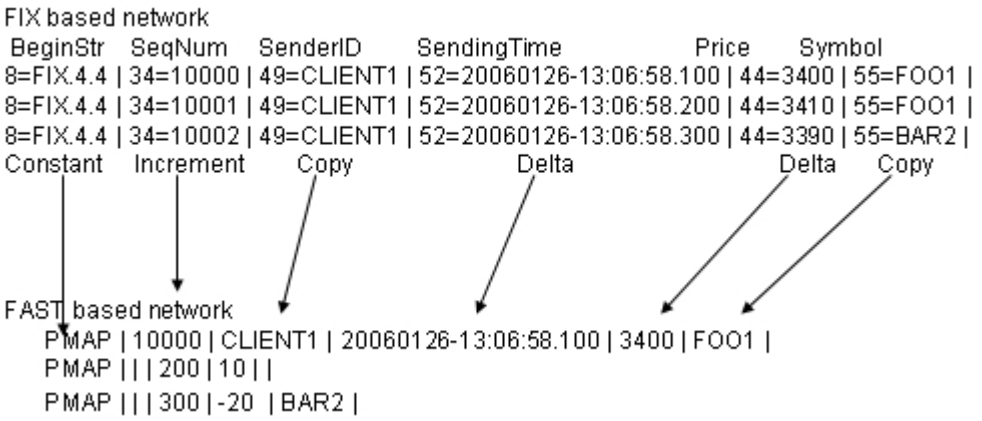

Figure 22. Several fields can be removed

In Figure 22, there are three FIX messages, starting 8=FIX.4.4. The first of them is sent almost in its full length, only data in Constant value field is removed, |1011111 | 10000 | CLIENT1 | 20060126-13:06:58.100 | 3400 | FOO1 |, this line goes to Transfer Encoding before it is sent to the receiver. The first message will give the first starting values for data fields on the templates at the receiver side, when same type of message is sent for the second time only the Delta values are sent |100110| | | 200 | 10 | |. The third message is slightly different because the company stock under exchange has changed from FOO1 to BAR2. The last bit in PMAP is now 1 due to there is data in the last field |100111| | | 300 | -20 | BAR2 |. From this simplified example can be seen the big reduction in data which is sent in the data cable. [6], p.35

5.6 Transfer Encoding

In the FAST compression method message goes first into Field Encoding where the excessive fields are removed. For the remaining fields Transfer Encoding is performed. In this chapter is shown in a bit level what takes place when Transfer Encoding is performed. In the example two messages are sent and both of them are identified to be same type therefore the Field Encoding can be performed and after this the Transfer Encoding is performed.

In Table 17 is shown each bit of one byte. This is to illustrate where the Stop bit is located.

Table 17. Bit order

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

In Table 17 is shown that bit 7 is reserved as a “stop-bit”, 6-0 are data bits. If the bit 7 is 1, it means it is the last byte of the field. One example is given, starting from Table 18, about byte and bit orders using templates, especially how the stop bit works. One template is illustrated in Table 18.

Table 18. Simplified template example

Field	Type	Operator
Message type	U32	COPY
Sequence number	U32	INCR
Timestamp	U32	DELTA
Freetext	U32	COPY

In Table 18 there is an example template. It defines the type of four data fields and an operator for each field. This template is on both sides, sender and receiver and as mentioned before, in front of this message there is the identifier field, recognition of the message is based on this. In this simplified example only the most viable information is shown, therefore the message identifier field is absent. Table 19 illustrates how two messages are sent using the same template.

Table 19. Content placed into the template above

Field	Message 1	Message 2
Field 1 - Message type	0x1234	0x1234
Field 2 - Sequence number	0x12345	0x12346
Field 3 - Timestamp	0x1234abcd	0x1234bbbb
Field 4 - Freetext	`Freetext`	`Freetext`

In Table 19 it is illustrated how two messages are using the same template. Operator type for each field from 1 to 4 can be seen from Table 18. One can see that in the template, Table 18, and in the message, Table 19, the data fields are in corresponding order to each other. In Figure 23 is presented one FIX message and this message is dissected into five parts and each part is then analyzed in the following section.

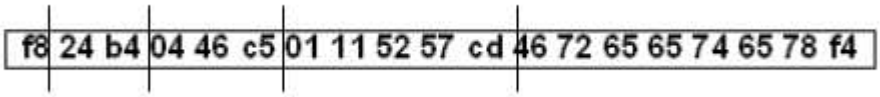

Figure 23. 5 separated fields

The easiest way to explain the usage of stop bits is by dissecting the encoded message. The line in Figure 23 is the encoded output of Message 1 from the Table 19, it is 19 bytes long.

The first byte in Figure 23 is the Presence Map PMAP. f8 is equal to 1111 1000 in binary form. The first bit is 1 and indicates that stop is set therefore this byte is the last byte of the field. Next four bits are set to be one 111 1, indicating that fields 1, 2, 3, 4 are present as marked to the example message and fields 5, 6, 7 are absent and those are marked 000. This is what is meant when it is said the FAST method is content aware, before it starts to encode or decode FAST already has knowledge of which fields are present and data type of those and correct operator for each field.

The second set of bytes in Figure 23 is 24 b4 and it is equal to 0010 0100 1011 0100 and these bits are interpreted in the following way, in the first byte the stop bit is not set, followed by 7 data bits. In the second byte the stop bit is set and followed by 7 data bits. The stop bit has no data carrying capabilities and it is not considered as a data bit. The 14 data bits are 01 0010 0011 0100 in hex form 0x1234. This is the original uncoded message. In a point when the message is encoded by FAST the stop bits are added. In the template the type of a message is defined and based on that stop bits are set.

The third set of bytes in Figure 23 is 04 46 c5 and it is equal to 0000 0100 0100 0110 1100 0101, and these bits are interpreted as stop bit not set, seven data bits, stop bit not set, seven data bits, stop bit set, seven data bits. The 21 data bits are 0 0001 0010 0011 0100 0101 in hex form 0x012345. In each byte the most significant bit is reserved for the stop bit and it reduces the number of data bits in this example down to 21 bits. Note that in each byte there is a stop bit and whether or not it is set it increases the total size of one byte by one bit.

The fourth set of bytes in Figure 23 is 01 11 52 57 cd and it is equal to 0000 0001 0001 0001 0101 0010 0101 0111 1100 1101, and these bits are interpret as stop bit not set, seven data bits, stop bit not set, seven data bits, stop bit not set, seven data bits, stop bit not set, seven data bits, stop bit set, seven data bits. The 35 data bits are 000 0001 0010 0011 0100 1010 1011 1100 1101 and in hex form 0x01234abcd.

The fifth set of bytes in Figure 23 is the “Freetext” without the brackets. In the last byte f4, stop bit is set, in all the other bytes stop bit is clear. The original message was 17 bytes long, after encoding the message is 19 bytes long. Adding one stop bit to each byte increases the size. Now the second message is analyzed. It is shorter than the first message, it is 11 bytes long. This is showed in Figure 24.

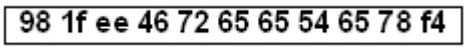

Figure 24. The second message

In Figure 24 there is the content of the second message. The original size of the message 1 and message 2 is the same but when the second message is transmitted, only 11 bytes are sent as in the first message there were 19 bytes. The second message is divided into three fields to analyze it.

The first byte is 98 which is the PMAP, 1001 1000, the first bit is 1 and therefore stop bit is set. Fields 1 and 2 are missing and those are marked to be 0. Fields 3 and 4 have data inside and those are marked by 1. Field 1 is marked by operator COPY therefore the data to field 1 is copied from the previous message on the receiver side and value of the INCRemental field 2 is incremented by one on the receiver side.

The second set of bytes is 1f ee and it is equal to 0001 1111 1110 1110, and this is interpreted as stop bit clear, seven data bits, stop bit set and seven data bits. The 14 data bits are 00 1111 1110 1110, that is in hex form 0x0fee which is the delta value of “Timestamp” 0x1234bbbb – 0x1234abcd = 0x0fee. In the template, Field 3 is defined to be Delta therefore delta value is calculated between Message 1 and Message 2 and only this delta value is sent.

The third set of bytes is the “FreeText” data. Here could read “Aikakone”. This kind of data cannot be compressed using templates. Field 4 requires 8 bytes just as in the first message. The second message is 11 bytes in size. [3]

In total two messages, 34 bytes, was reduced down to 28 bytes. When two messages are sent the compression rate is 18%, when three messages are sent 17+17+17 = 51 bytes is compressed to 17+11+11 = 39 bytes which means a 24% reduction in data. These messages, see Table 19, are not realistic in real trading but are usable examples to show FAST compression in the bit level and the usage of stop bits.

In Figure 25 is shown FIX messages to illustrate how data fields are removed.

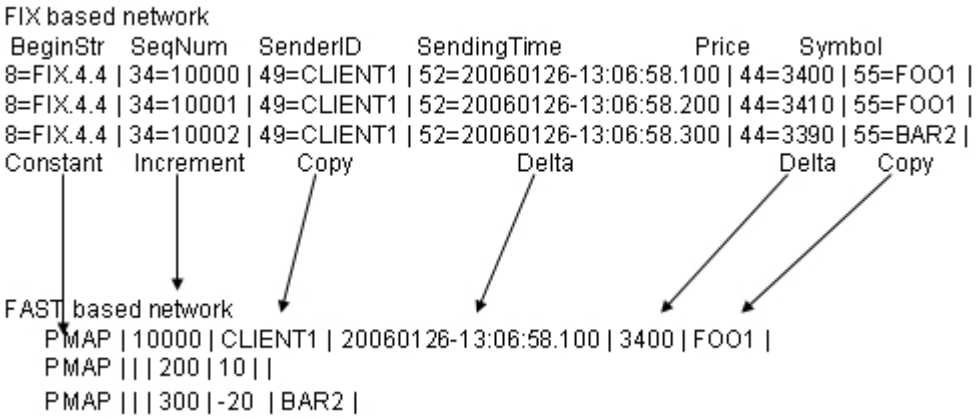

Figure 25. FAST method reduces fields witch are sent

Figure 25 shows how three FIX messages each in length of 71 bytes are sent. The first message is starting 8=FIX.4.4 and it is 71 bytes in length, in the message one byte is reserved for each mark. The Field Encoding reduces the size of the first message down to 47 bytes+PMAP 1 byte, reduction is 33%. The second message is 11 bytes+PMAP 1 byte, reduction is 83%. The third message is bigger because traded stock has changed, 16 bytes+PMAP 1byte, reduction is 76%. By using templates and field operators several fields can be removed.

5.7 Genium Bandwidth Calculations

This chapter covers the bandwidth calculations for Genium. The same study is used as in the Saxess calculations to find the ratio between the three most common messages and the trade. Saxess uses four messages to perform most of the actions in the trading network but in Genium there is only one message but the ratio is needed for calculations.

In Table 20 is presented the number of messages in each message type.

Table 20. Most common messages in Saxess

Message type	Number of messages	Message distribution	Total size in bytes	Size distribution
MBO OrderInsert	325 365	41,2%	41 321 355	44,4%
MBO OrderCancel	233 962	29,7%	17 079 226	18,4%
Trade	98 412	12,5%	16 927 224	18,2%
MBO OrderUpdate	48 081	6,1%	7 019 826	7,5%

From the study of Tomas Lundqvist and Rolf Andersson this Table 20 is used to analyze how many messages there are in respect to one trade. [3]

325 365 + 233 962 + 48 081 = 607 408 messages
607 408 / 98 412 = 6,17 messages per on trade.

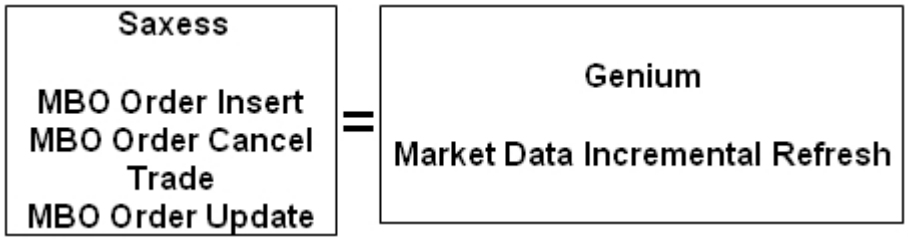

Figure 26. Most common trading messages in Saxess and Genium

In Genium, Market Data Incremental Refresh message takes care of the same functions as those four messages in Saxess.

Market Data Incremental Refresh containing Order Book Updates is 222 bytes [8] in size. This message handles the same functions as MBO_Order_Update, MBO_Order_Cancel and MBO_Order_Update in Saxess.

Market Data Incremental Refresh containing Trade Data is 310 bytes [8] in size. This message handles the same function as the Trade message in Saxess.

Now is calculated an average of how much traffic one trade generates in Genium.

6,17 * 222 bytes = 1370 bytes for messages other than trade
1 * 310 bytes = 310 bytes for a trade message

1370 + 310 = 1680 bytes. One trade generates this much traffic

FAST compression reduces it down to 0,15 * 1680 = 252 bytes [8]

252 bytes * 105 trades = 26 460 bytes

26 460 * 8 = 211 680 kbps

WAN frames

To compare to Saxess, 105 trades per second is used for calculations.

105 * 6,17 = 648 pcs. of other than trade messages.

105 trade messages

648 + 105 = 753 messages altogether to execute 105 trades in Genium.

Max 4 FIX messages in the same TCP segment

753 / 4 = 189 TCP segments

Size of one WAN frame 66 bytes

66 * 189 = 12 408 bytes [8]

Genium bandwidth after compression and WAN frames

26 460 + 12 408 = 38 868 bytes / sec.

38 686 * 8 = 310 944 bps [8]

105 trades in Saxess 450 kbps per member server, or 7x105=735 messages

105 trades in Genium 311 kbps per member server, or 7x105=735 messages. In Figure 27 is presented how much traffic is been generated on four different levels of trades.

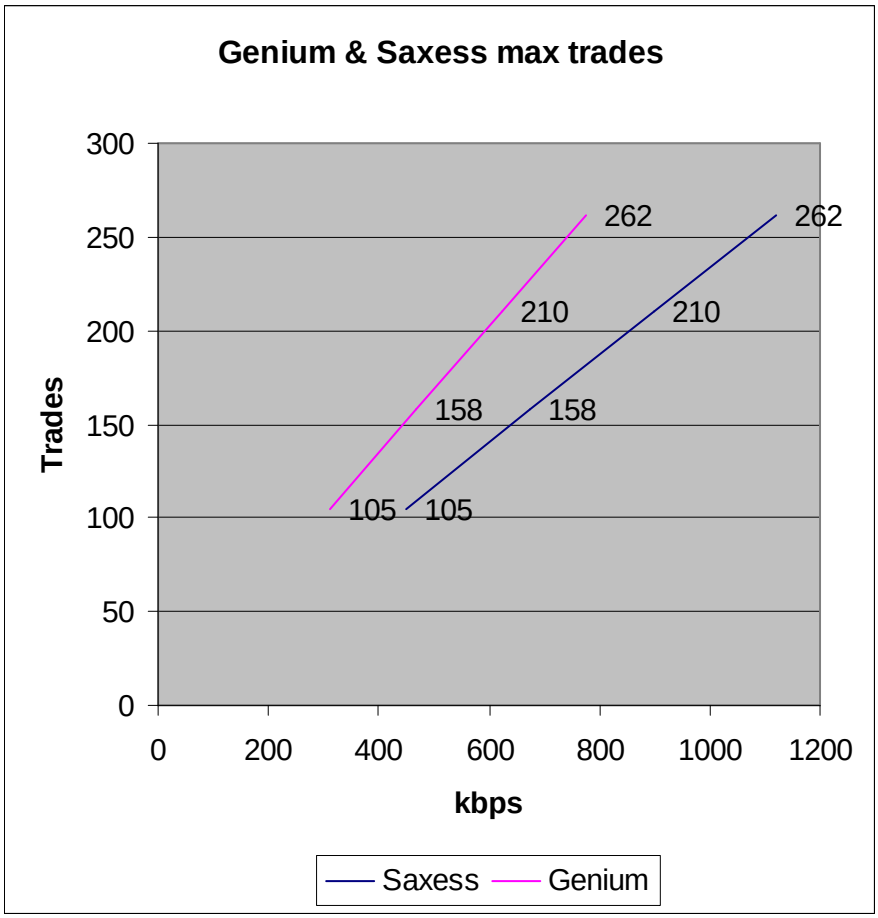

Figure 27. Illustrates number of trades in Saxess and Genium

In Figure 27 can be seen that Genium consumes less bandwidth than Saxess when the same number of trades are executed. The results of the

study presented in Figure 27 are the values which can be calculated fairly easily. In Saxess there are four messages which create 88,5% of the bandwidth load. Into this 88,5% bandwidth has to be added 11,5% extra, then it includes all the least common messages such as MBL Market By Level messages and other less common messages. For the Genium bandwidth calculations were used the ratio from the Saxess network therefore also Genium bandwidth calculations have to be increased by 11,5%.

6 DISCUSSION AND CONCLUSIONS

In this study the bandwidth load was calculated by analyzing the four most common messages. These four messages generate most of the bandwidth load. One has to add some extra to the bandwidth because there are hundreds of other messages which create the minority of the network load. These less common messages are not analyzed in any detail because it would be too big work load to do it. The object of this study was to find out if there was a need to change bandwidth configurations for members when changing into new trading platform. As the study shows, the new Genium will use less bandwidth than the existing trading platform Saxess. One could think to buy less bandwidth from the network operator when Genium is taken into use but it is not feasible, because after all, Genium requires only about 100 kbps per server less than Saxess and with the current rate how incoming orders are raising, this 100 kbps extra would be soon needed. Therefore, there is no need to change bandwidth configurations.

In the bandwidth calculations the average number of how many messages there were in respect to one trade was calculated. The final results are not absolute because members are free to send sell and buy orders in as many as they want. In some points the bandwidth can be in its peak with only 60% of trades calculated in this study being executed. This might happen if there were many orders coming in but orders are not matched. A more detailed study could be conducted about how the traffic alters.

REFERENCES

[1] OMX Technology AB, XTP 2.56 Specification 14.8.2007
http://omxnordicexchange.com/memberlogin/saxess/developer/

[2] OMX Technology, 7466_Saxess_AccessProtocol_SFN.pdf 20.9.2007
http://www.omxnordicexchange.com/memberlogin/saxess/communication/

[3] Tomas Lundqvist & Rolf Andersson, OMX_FASTpaper_051029 [email]

[4] OM Gruppen AB, XMP Exchange Message Protocol 13.8.2008
http://www.oslobors.no/servlet/BlobServer?blobtable=Document&blobheader=application%2Fpdf&blobwhere=1043933654374&blobcol=urlblob&blobkey=id&1043933654374.pdf

[5] Matt Simpson, CME, 08_Basic_FAST:Implementation – Simpson 2.3.2007
http://www.fixprotocol.org/documents/2525/FAST%20Tech%20Summit%20-%20CME.zip

[6] Kevin Houston, HSBC Investment Bank, FAST_Breakfast_Briefing_kh 15.2.2008
http://www.fixprotocol.org/documents/2701/FAST_Breakfast_Briefing_kh.pdf

[7] Richard Gaudy, Nasdaq OMX, 29.8.2007, Trading Members (Cash Market) traffic forecast [email]

[8] Per-Anders Sahlin, Nasdaq OMX, 14.3.2008
GMI_FIX_message_bandwidth [email]

[9] Benny Rachlevsky-Reich, A Global Electronic Market System, available at www.sciencedirect.com, Published by Elsevier Science Ltd.

[10] M. Mandjes, Bandwidth trading under misaligned objectives, available at www.sciencedirect.com, Published by Elsevier Science Ltd.

[11] OMX, Opi osakkeet, available at Nasdaq OMX Fabianinkatu 14 Helsinki Customer Services

Now is calculated how much traffic one trade generates in average in Genium.
6,17 * 222 bytes = 1370 bytes for messages other than trade
1 * 310 bytes = 310 bytes for trade message

1370 + 310 = 1680 bytes. One trade generates this much traffic
FAST compression reduces it down to 0,15 * 1680 = 252 bytes [8]
252 bytes * 105 trades = 26 460 bytes

26 460 * 8 = 211 680 kbps

WAN frames
To compare to Saxess, 105 trades per second is used for calculations.
105 * 6,17 = 648 pcs. of other than trade messages.
105 trade messages
648 + 105 = 753 messages altogether to execute 105 trades in Genium.

Max 4 FIX messages in the same TCP segment
753 / 4 = 189 TCP segments
Size of one WAN frame 66 bytes
66 * 189 = 12 408 bytes [8]

Genium bandwidth after compression and WAN frames
26 460 + 12 408 = 38 868 bytes / sec.
38 686 * 8 = 310 944 bps [8]

105 trades in Genium 311 kbps per member server

Now is calculated how much traffic one trade generates in average in Genium.
6,17 * 222 bytes = 1370 bytes for messages other than trade
1 * 310 bytes = 310 bytes for trade message

1370 + 310 = 1680 bytes. One trade generates this much traffic
FAST compression reduces it down to 0,15 * 1680 = 252 bytes [8]
252 bytes * 158 trades = 39 816 bytes

39 816 * 8 = 318 528 kbps

WAN frames
To compare to Saxess, 158 trades per second is used for calculations.
158 * 6,17 = 975 pcs. of other than trade messages.
158 trade messages
975 + 158 = 1133 messages altogether to execute 158 trades in Genium.

Max 4 FIX messages in the same TCP segment
1133 / 4 = 284 TCP segments
Size of one WAN frame 66 bytes
66 * 284 = bytes [8]

Genium bandwidth after compression and WAN frames

39 816 + 18 744 = 58 560 bytes / sec.
58 560 * 8 = 468 480 bps [8]

158 trades in Genium 469 kbps per member server

Now is calculated how much traffic one trade generates in average in Genium.
6,17 * 222 bytes = 1370 bytes for messages other than trade
1 * 310 bytes = 310 bytes for trade message

1370 + 310 = 1680 bytes. One trade generates this much traffic
FAST compression reduces it down to 0,15 * 1680 = 252 bytes [8]
252 bytes * 210 trades = 52 920 bytes

52 920 * 8 = 423 360 kbps

WAN frames
To compare to Saxess, 210 trades per second is used for calculations.
210 * 6,17 = 1296 pcs. of other than trade messages.
210 trade messages
1296 + 210 = 1506 messages altogether to execute 210 trades in Genium.

Max 4 FIX messages in the same TCP segment
1506 / 4 = 377 TCP segments
Size of one WAN frame 66 bytes
66 * 377 = 24 882 bytes [8]

Genium bandwidth after compression and WAN frames
52 920 + 24 882 = 77 802 bytes / sec.
77 802 * 8 = 622 416 bps [8]

210 trades in Genium 623 kbps per member server

Now is calculated how much traffic one trade generates in average in Genium.
6,17 * 222 bytes = 1370 bytes for messages other than trade
1 * 310 bytes = 310 bytes for trade message

1370 + 310 = 1680 bytes. One trade generates this much traffic
FAST compression reduces it down to 0,15 * 1680 = 252 bytes [8]
252 bytes * 262 trades = 66 024 bytes

66 024 * 8 = 528 192 kbps

WAN frames
To compare to Saxess, 262 trades per second is used for calculations.
262 * 6,17 = 1617 pcs. of other than trade messages.
262 trade messages
1617 + 262 = 1879 messages altogether to execute 262 trades in Genium.

Max 4 FIX messages in the same TCP segment
1879 / 4 = 470 TCP segments
Size of one WAN frame 66 bytes

66 * 470 = 31 020 bytes [8]

Genium bandwidth after compression and WAN frames
66 024 + 31 020 = 97 044 bytes / sec.
97 044 * 8 = 776 352 bps [8]

262 trades in Genium 777 kbps per member server