Final Report

(IPRO313 - Ultra-High-Speed Market Data Ticker System)

Sponsor: Townsend Analytics

Manager: Carl Ververs

Illinois Institute of Technology

Advisors: Wai Gen Yee

Ben Van Vliet

	Design Team:	**Hardware Team:**	**Software Team:**
Students:	Philip Pannenko	Michael Lenzen	Jongyon Kim
	Devaraj Ramsamy	Yunseok Song	Young Cho
	Kenneth Buddell	Jong Min Lim	Usman Jafarey
			Jong Su Yoon
			Jesus Allan C Tugade

Version #	**1.0**
Filename	IPRO313_IPRO Final Report_20071130_v1.0

TABLE OF CONTENTS

INDEX OF VISUAL AIDS

1. NTRODUCTION

1.1. Background

With the dawn of every New Year, the speed of business is ever increasing. What used to be performed in months, days or minutes is now being done in milliseconds; and even that isn't fast enough! For businesses to stay competitive within their industries, many will need to transition to real time data management. Infrastructures will need to be vastly renovated and millions of dollars will be put into preparing for the huge amounts of data going through the system. The question is: will it be worth the hassle?

[Figure 1: Benefits of Real Time Data Integration]

Better decisions:
With a more dynamic view of information as it flows in and out of the business, decision makers can better understand business processes than if the information is provided in time-lagged reports

Faster response to material events:
Immediate awareness of changes to acceptable business conditions or to the presence of material events removes latency from business processes for a more agile enterprise.

Unwired operations:
The ability to alert and notify key individuals to changes within the business enables greater flexibility in where business can happen.

Proactively publish data at lower cost and less complexity:
Move data events immediately from multiple databases directly to a messaging infrastructure without the cost of custom coding or changing existing applications.

Preserve operational performance:
When capturing events from database systems, there is zero impact on the source database, preserving operational performance without incurring additional system overhead costs.

Complement existing messaging infrastructures:
Captured events are standardized and distributed to business applications throughout the enterprise via a standards-based messaging infrastructure without coding custom interfaces.

Manage information flows with flexibility:
Sybase Real-Time Data Services facilitates how customers can choose to flow data events through the organization—such as enriching events with analytical or historical information before arriving to the decision-maker—without slowing down operational system performance.

Source: Sybase Real–Time Data Services [10]

Businesses in the telecommunications, military, medical, energy, and financial industries benefit from real time data management:

[Figure 2: Examples of Real Time Data Management Integration Benefits]

Industry:	Real Time Integration Benefit:
Telecommunication	Dispersing calls automatically on a network to guarantee completion of connections
Military	Simulating test environments on real time threats
Medical	Health monitoring systems alerting hospital personal faster of patient condition
Energy	Predicting and preventing power shortages based on current surges and spikes
Financial	"Placing a buy or sell order of a defined quantity into a quantitative model that automatically generates the timing of orders and the size of orders based on goals specified by the parameters and constraints of the algorithm." [3]

1.1.1. Financial Industry – Quick Intro and Classical Approach

"Buying and selling shares of stock is at the root of American capitalism, the mechanism by which great companies have been built ever since the New York Stock Exchange was created way back in 1789." [1]

Trading is the voluntary exchange of goods or service and it exists due to regional comparative advantages and specialization/divisions of labor. Trades are based on an auction market paradigm where a potential buyer bids a specific price and a potential seller asks a specific price:

[Figure 3: Classic Trade Flow]

Step 1: Decision
Portfolio manager uses research, fundamentals and historical market data to make investment decision

Step 2: Place Order
Buy-side trader calls the sell-side trader and communicates trade over the phone

Step 3: Interact with market
Sell-side interacts with the marketplace or takes on position

Step 4: Relay Trade
Sell-side calls buy-side trader with execution information.

Step 5: Enter in Ledger
Buy-side books the trade on the position ledger.

Source: Managing Risk in Real-Time Markets [10]

1.1.2. Introduction of Electronic Trading

In 1975, there was request to encourage “maximum reliance on computer and communications technologies.” [2] The reason for this was to eliminate as much of the redundant paper work that slowed the trading process down as possible. The implementation of this however also resulted in considerably faster access to market data, by the millions!

What can anyone possibly do with information arriving in such quantities and speeds on these “fire hose” data feeds? The obvious answer is that people alone cannot do anything with that much information. The receiving end always involves a computer. The kinds of tasks we can do with computers have grown so incredibly that, to use them effectively, we have to change the way we think about them. [2]

By severely reducing the amount of human intervention, electronic trading has radically changed the nature of the classical trade flow.

[Figure 4: Electronic Trading Vamps Trading Flow]

Step 1: Decision
Computer programs dubbed “Black boxes” analyze data and derive buy or sell requests

Step 2: Place Order
Direct Market Access (DMA) allows the computer to place an order directly with a vendor

Step 3: Interact with market
A Volume Weighted Average Price (VWAP) algorithms facilitate efficient execution of large orders

Step 4 & 5: Order and Trade Management
Portfolios are updated and managed by the computer

Source: Managing Risk in Real-Time Markets [10]

1.1.3. Trading Is Automated So Where Is the Hassle?

Automated trading is, automated. Within one second between 5,000 and 50,000 changes can occur [11]. That means that within one year, terabytes of data will need to be handled by specially prepared warehouses full of zero-latency databases. A sudden transition of an ordinary warehouse of database to the different style of processing required, data volume, velocity, and rate of change required for real time processing would certainly cause a database overload. Gradual shifts need to be made for a healthy implementation.

[Figure 5: 6 Steps to Real Time]

Employ Complex Event Processing (CEP)
CEP is technology designed to process multiple streams of simple data events with the goal of identifying the meaningful events within those streams.

Cache data In-Memory
Simply analyzing raw event data is useless without access to reference data that helps inform decisions.

Exploit Distributed Shared Memory Grids (DSMG)
DSMG allow large sets of data to be shared and distributed, so event processing can proceed at in-memory speeds for real time automated trading decisions to be made. Integrated with a database, shared memory grids enable real time data access.

Implement EDA & SOA - at the Same Time
Event-driven architecture can be service-oriented, and vice-versa

Be Analytical, Be Curious, and Be Bold
Real time is bold change that can create dinosaurs out of a business's competition.

Iterate, Embrace & Extend
Only about 20 percent of the world's existing IT systems will become real time, so plan for real time systems to live with old time systems for a long time.

Source: The Real Time Data Management Imperative [11]

1.1.4. Townsend Analytics

Townsend Analytics (TAL), a direct-access trading-system vendor, provides connectivity to a multitude of electronic-communications networks and stock exchanges:

Servicing the global capital markets for over 20 years, Townsend Analytics Trading Services offers world class trading and trade order management solutions to the institutional- portfolio management and broker dealer- marketplace. In today's competitive institutional trading environment, portfolio managers and traders are under increasing pressure to access a wider variety of liquidity sources, employ more aggressive yet cost efficient trading strategies and achieve best execution faster than ever. Achieving this demands the right service bureau partner, powerful order management/reporting tools and direct access to a variety of electronic markets across multiple asset classes. [4]

TAL has already developed a means for managing real time data which has captured both the Unites States market as well as the European market:

Powered by RealTick®, Townsend Analytics' flagship institutional product, RealTick EMS (Execution Management System) is the institutional financial service industry's leading multi-asset, multi-broker, multi-routing and multi-regional market data, analytics and direct market access (DMA) trading platform. [5]

1.2. Electronic Trading Market

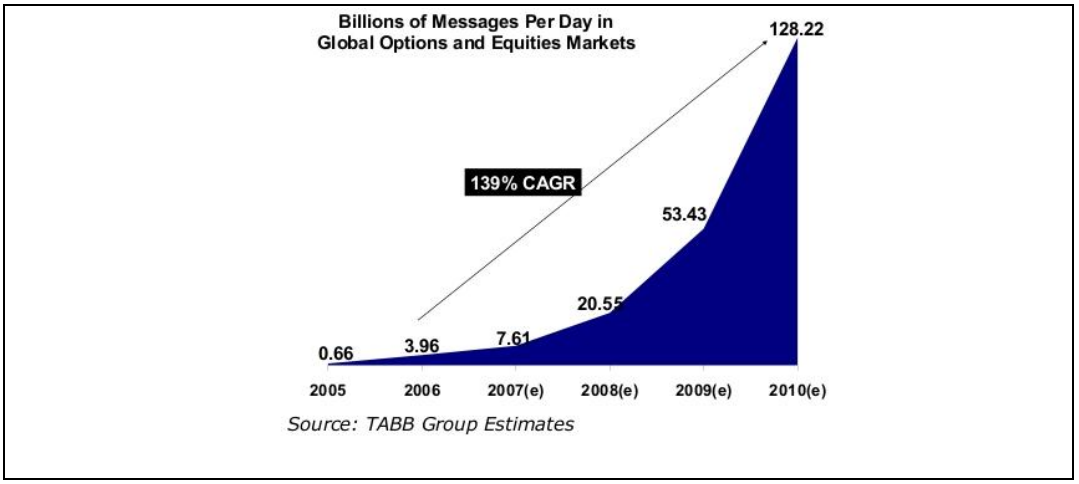
1.2.1. Current Market Competitors

Currently there are a number of companies that address this same problem; Exegy, Reuters, RMDI, Wombat are examples of such. These companies however lack an effective system that will aggregate the amount of expected data in the next few years. Comparisons of these competitors' solutions are included in section 3.2.2.

1.2.2. Market Value

According to TowerGroup, a research firm, $480m is likely to be spent in America this year on developing technology for algorithmic trading. In 2005, providing fastest data to customers had such an impact that the financial industry increased spending on computers and software to $26.4 billion [7] and in the past years, the compound annual growth rate for algorithm use from 2004 through 2007 was projected at 34%. [8] Such is the focus on speed that even location counts. Servers positioned nearest to a trading venue can shave milliseconds of the timing of a trade and get a better price. [6]

[Figure 6: Predicted Increase in Messages per Day]



Source: Trading At Light Speed: Analyzing Low Latency Data Market Data Infrastructure [4]

Ultimately, because the need for speed of information will only continue to grow, if TAL can develop a unique and reliable solution, it will provide a vital product for a demanding future market.

1.3. Purpose

1.3.1. Problem Addressed

IPRO 313's objective is to develop a data ticker plant for our sponsor Townsend Analytics which needs to meet or exceed certain performance requirements. The data ticker plant has to have a sustained optimal throughput of three million price quotes per second and minimize latency while maintain specific constraints. The ticker plant aggregates streaming data for numerous global financial markets and disseminates the data to thousands of users in real time. The data is used in Townsend Analytics' RealTick® Execution Management System (EMS), its flagship institutional product for the financial services industry. Thus, timely and accurate data delivery is a critical component to Townsend's product and competitive position. Through research and development, the group will have a concept of design, prototype development and benchmark testing. Additionally, the groundwork will be laid for future development of the ticker plant and additional trading-platform components.

1.3.2. Objectives

The group prioritized their objectives as follows:

- Research low latency discussions and reports
    - Fully understand what a ticker plant is and does
    - Learn about new methods for ticker plants and explore their advantages and disadvantages
    - Understand market use
- Explore competitors' solutions
    - Know what is currently on the market
    - Better understand implementation of ticker plant architecture
    - Understand what works and what does not
- Develop a functioning ticker plant system
    - Analyze ticker protocols used
    - Design ticker plant architecture
    - Code a working small system
- Determine hardware requirements
    - Test off-the-shelf hardware for system
    - Design custom hardware configuration
    - Compare each solution
- Benchmarks & Prototype

- Integrate hardware and software designs
- Prepare benchmarks
- Document technical user manual

1.3.3. Expected deliverables:

- Design (Ticker Plant System)
- Comprehensive List of Research Articles
- Extensive Competitive Solution Documentation
- Source Codes for Prototype of the Ticker Plant System
- Benchmarks / Test Results
- Documentation
- Technical Manual
- User Manual
- Guidelines for potential future IPRO's

2. METHODOLOGY

The IPRO group is broken down into three teams: Design, Hardware and Software. Each team is responsible for researching and developing solutions within its specific area, but also informs and collaborates with the other teams of their findings. Each team will draft its own reports, schedule and presentations. Additionally, each team will report to the group weekly, so that the information is presented in a timely and consistent manner. With a weekly schedule of presentations and reports, the flow of information is communicated to everyone on the project and the oversight of work is being maintained. Documents generated from each group are to support the IPRO deliverables, so that they can be streamlined into a comprehensive deliverable.

2.1. Design:

The Design team is responsible for all IPRO deliverables and deadlines and for managing the work flow of the entire project. It is also responsible for communicating weekly with the sponsor on the progress of work done for the past week and for upcoming work. The design team, aside from taking a managerial role, will conduct research on the financial industry. Research topics include competitors, regulations, protocols, and overall market conditions now and expected future market conditions.

2.2. Hardware:

The Hardware team will determine whether standard off-the-shelf hardware will meet the requirements both now and in the future. The group will justify their decision with research into the growth in hardware capabilities and analysis of different current hardware configurations through experimentation. If the case is that standard servers will not suffice we will come up with a proposal for a solution that will work and justify why we think that it will work the best. The hardware design encompasses more than just the design of each machine, but also the overall design of the system. The group will also analyze competing products, determining what they are using and why.

Work throughout the semester will include:

- Creation of tests and benchmarks to simulate demand on the system
- Test simultaneous network I/O by testing the maximum throughput while varying message sizes and ratio's of input to output
- Testing of data access times and data updates
- Coordinate work with software team in development of the macro-design

2.3. Software

The software team will focus on Data Streaming and Last Value Cache in the processing system over this semester with methods via research into relevant paper from academic side and benchmark other systems from industrial side. The group will propose reliable solutions and implement programs with statistical data.

2.4. Division of Tasks

2.4.1. Schedule of Tasks & events

Task Name	Time	Start	Finish	Resources
Research & Developing				
Research on Different Architecture	2 Weeks	1-Oct	13-Oct	Software
Designing the Ideal Architecture	2 Weeks	1-Oct	13-Oct	Software
Competitors Documentation	10 Days	16-Sept	26-Sept	Hardware
Annotating White Papers	1 Week	6-Oct	13-Oct	Design
Market Terms/Value	2 Week	1-Oct	13-Oct	Design
IPRO Midterm Report				

Midterm Report	1 Week	10-Oct	26-Oct	Design
Ethics Paper	1 Week	10-Oct	17-Oct	Design
Oral Report	1 Week	10-Oct	17-Oct	Design
Test				
Determining Test Case Format	1 Week	8-Oct	15-Oct	Design
Module Performance	2 Weeks	15-Oct	27-Oct	Software
Modify and Optimizing the Source Code	2 Weeks	15-Oct	27-Oct	Software
Simultaneous Network I/O	2 Weeks	27-Sept	9-Oct	Hardware
Data Access Times & Updates	2 Weeks	27-Sept	9-Oct	Hardware
Analysis & Improvements				
Designing Guidelines For Next Phase	1 Week	27-Oct	5-Nov	Software
Proposing of New Features	1 Week	27-Oct	5-Nov	Design
Multiple Network Cards	2 Week	11-Oct	29-Oct	Hardware
Integration				
Components Tested As Whole	3 Weeks	29-Oct	17-Nov	ALL
Error Debugging & Correction	3 Weeks	29-Oct	17-Nov	ALL
IPRO Deliverables				
Poster	2 Weeks	17-Nov	30-Nov	Design
Final Report	2 Weeks	17-Nov	30-Nov	Design
Presentation	2 Weeks	17-Nov	30-Nov	ALL
Technical Write Up	2 Weeks	17-Nov	30-Nov	Hardware
User Manual	2 Weeks	17-Nov	30-Nov	Software
Industry Report	2 Weeks	17-Nov	30-Nov	Design
Benchmarks	2 Weeks	17-Nov	30-Nov	Hardware

2.4.2. Individual Assignments

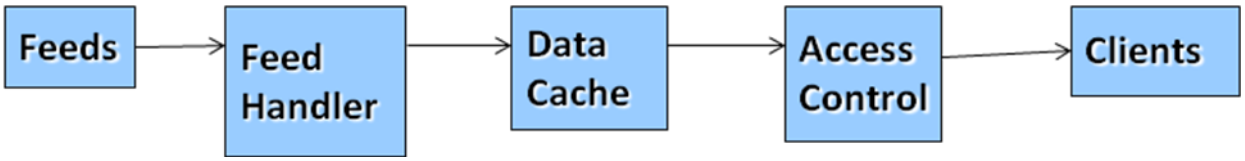
Group Members	Major	Skill & Strengths	Experience	Roles
Design Team				
Philip Pannenko	Computer Science	Management & Communication	CBOE employment	IPRO Team Leader; Agenda Maker
Devaraj Ramsamy	Business	Project Controls	Bechtel SAIC LLC, DOE Yucca Mountain Project	iGROUPS & Deliverable Management
Kenneth Buddell	Business		Previous IPRO experience	Timesheet Summarizer Industry Research
Software Team				
Jongyon Kim	Business	Management	IT Consulting	IPRO Sub Team Leader; Minute Taker; Timesheet Summarizer
Young Cho	Applied Math	C++	Research	Research, Programmer
Usman Jafarey	Computer Science	C++	Data streaming project	Programmer
Jesus Allan C Tugade	Computer Science	JAVA	Navistar IT development group	Programmer
Jong su Toon	Computer Science	C, C++, Java, Oracle, PHP	Online Game Company	Software Coding
Hardware Team				
Michael Lenzen	Applied Math	Java, JavaScript, PHP, Perl	Web start-up company	IPRO Sub Team Leader; Timesheet Summarizer
Yunseok Song	Electrical Engineering	C, Matlab, Circuit Analysis	N/A	Research
Jong Min Lim	Electrical Engineering	Statistical Analysis, Circuit Design	Power Analysis Com Ed	Network Analyst

2.5. Project Budget:

Name	Description	Price
IPRO Items		
Paper	Brochure, Abstract, Summary Sheets	$20
Poster	Poster board/printing	$30
Printing	Color printing	$40
TAL Meetings	Lunches with sponsor	$100
	Total	$190

3. Survey of System

3.1. Basic Architecture

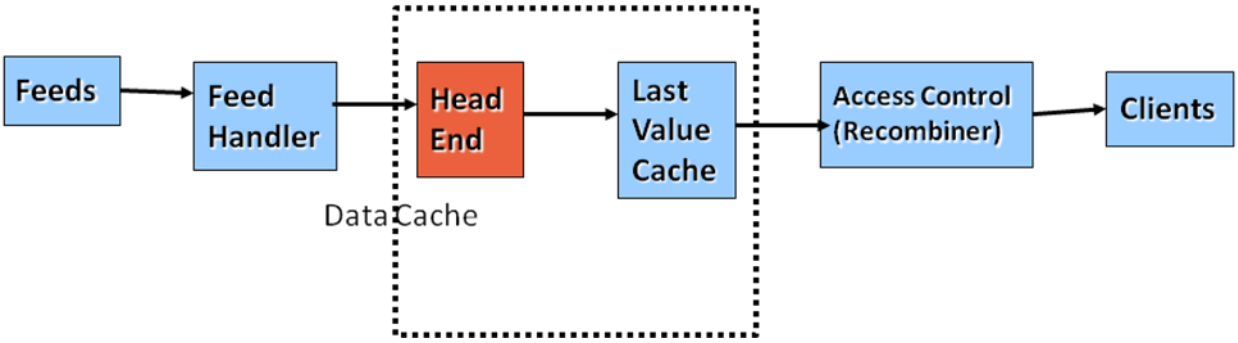


[Figure 7: Generic Architecture]

Basic architecture of the ticker plant system is derived based on the common factor of competing systems.

- Feeds: Options, Stocks, Futures
- Feed Handler: Translates incoming data to an internal format
- Data Cache: Stores old data
- Access Control: Handles client connections and permissions
- Client: Requests and receives ticks through the system

3.2. Townsend Analytics (TA)



[Figure 8: Townsend Analytics' Architecture]

3.2.1. Performance Number

1M messages per second

3.2.2. Market

Townsend Analytics currently has 5000 users (Institutional 50% / Client 50%). Since one CPU cannot handle all the transaction that occurs during the process, TA is using 16 quad-core processors (400K messages per second).

3.2.3. Features

- Feed Handler translates all incoming messages to TAL4
- Head End normalizes data by filling in missing values from last value cache
- Their data cache only stores the last value
- The recombiner acts as access control
- Subscription information is kept
- Data requests are handled and sent to the last value cache
- Clients use Realtick (software) to receive the data

Realtick is software developed by Townsend Analytics. It is used as a consolidator. It offers data analysis and feed data both of which clients can subscribe to. For example, Realtick trading platform has Global Equity and Algorithmic Trading which is developed by Merrill Lynch.

3.2.3.1. Feed Handler

Receiving Feeds from Exchanges and translating the format to TAL4 which is the format Townsend is currently using.

3.2.3.2. Cache server (Head End / Cache Server in TA)

Normalizing process will be the part where all the data is hashed into appropriate location of an array. Partial data received from the data source will then be normalized by filling in the blanks with values from the last value cache, update the last value cache and distribute the data to the clients with all the blanks filled in. When there are two different values for one symbol, it will be caching only the latest value. So, when the end users boot up the Realtick, they will be accessed with the full packet, and from then onward, they will be continuously updated with any recent value that they are subscribing.

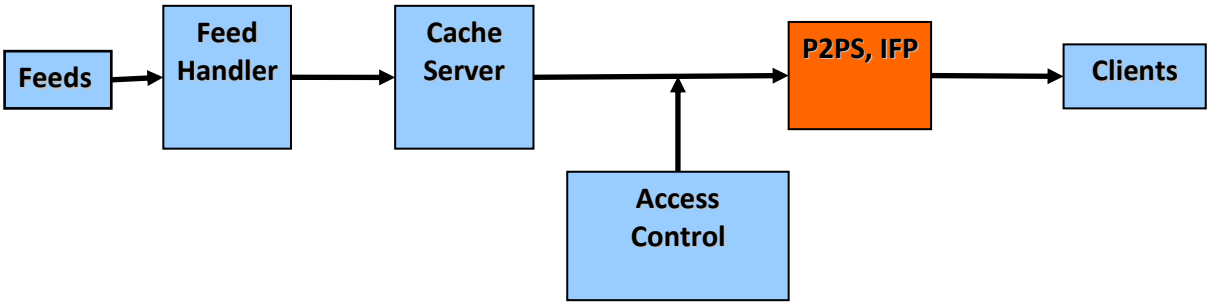
3.2.3.3. Recombiner (Access Control)

This is the component where subscription is kept and will distribute the data packet to their clients according to their subscription. All the data before here will be access through the internal process of TA and after here, it will be sent out from TA through the Internet service.

3.2.4. Hardware Specification

- Using 16 quad-core processors (400K messages per second)
- 16 Feed Handler
- 16 Head End* three for backup purpose (staging, deployment...)
- 32 Cache Server

3.3. Reuters Market Data System (RMDS)



[Figure 9: Reuters Market Data System's Architecture]

3.3.1. Performance Number

- Dual core 3.0 GHz / Novell Linux server / 1G-bit Ethernet
- 1 millisecond or less of end-to-end infrastructure latency at up to 300,000 updates per sec
- 2.59 million outbound user updates per second in the Point-to-Point Server fanout test
- Quad core 2.93GHz / Intel "Caneland" server / 1G-bit Ethernet
- 2,800,000 ups through a single Source Distributor machine
- 2,200,000 updates per second through a single Point-to-Point Server machine

3.3.2. Market

- Approximately 16,900 staff in 94 countries
- Around 370,000 individual users

- Information on 38,000 companies worldwide
- Real-time data provided on 5.5 million financial records
- Financial information from over 160 exchanges and OTC markets

3.3.3. Features

RMDS makes financial trading software and hardware. The system is consist of Feed handler (RDF, Third party source distributor, RDF +, RFA C++ source), Source distributor, Market data hub (RRCP), P2PS, Client's software.

3.3.3.1. Feed Handler

Reuter Data Feed (RDF): RDF is one type of feed handler and it makes it possible to provide full market data across all asset classes. Data are converted some kinds format such as RWF (Reuter Wire Format) or OMM (Open Message Model) to use data in RMDS at this level.

Reuter Data Feed Direct (RDF Direct): This feed hander is to make it faster to deliver the data to clients by removing middle procedures and converting data to certain format.

3.3.3.2. Cache Server

Source distributor: Source distributor receives data (which are converted in certain types of format (i.e. RWF, OMM)) from RDF, Third party sources, RFA C++ sources, and RDF Direct. The main function of source distributor is to implement source features such as recovery, data quality, load balancing, resource management, and support interactive publishers. Also, it stores last value cache to satisfy new requests for current items.

3.3.3.3. Market Data Hub (MDH): Communication thruway

The following three components make RMDS different from the other systems in structural point of view. They divide what access controller does in generic architecture into three components so that they can provide different data service according to what their client want.

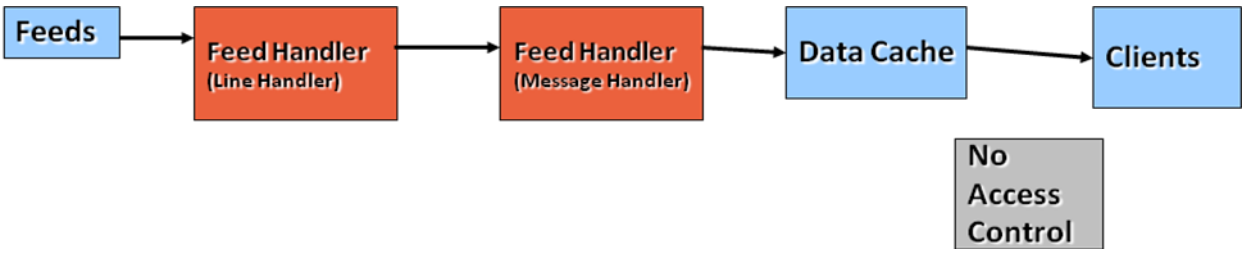
The Reuters Internet Finance Platform (IFP): RMDS usually uses their own connection, but this platform enables clients to connect to MDH and see the data served through RMDS by using IP network with Netscape or Explorer.

3.3.3.4. Access control

Data Access Control System (DACS): It is a system that enables client to control real-time data access to information from vendors, exchanges, specialist data services, and internally generated data. This system controls and supervises MDH so that data could be transmitted to clients properly according to their purpose or request types.

Point-to-point server (P2PS): It provides point to point access to all the information in market data hub and optionally caches it to redistribute to clients who request it. It also stores the information about point to point servers to reconnect easily and quickly. And, it converts RDM (I think it is Reuter Data Module)/RWF in the MarketPrice domain to Marketfeed for client applications. It also converts Marketfeed data to RDM/RWF so that RFA applications do not need to inspect Marketfeed.

3.4. Wombat



[Figure 10: Wombat's Architecture]

3.4.1. Performance Number

Latency of between 0.2 and 0.4 ms

3.4.2. Market

Wombat provides a software platform to handle direct market data feeds. This allows/forces a client to get direct feeds from the markets that they are interested in. The downside to this is that the clients do not get the benefit of having all of the data aggregated by a 3rd party. The benefits are that they only receive the data that they are interested in, but more importantly that the intermediary is cut out so the speed is improved. Because they are only a software solution, the client is also responsible for the hardware that it runs on.

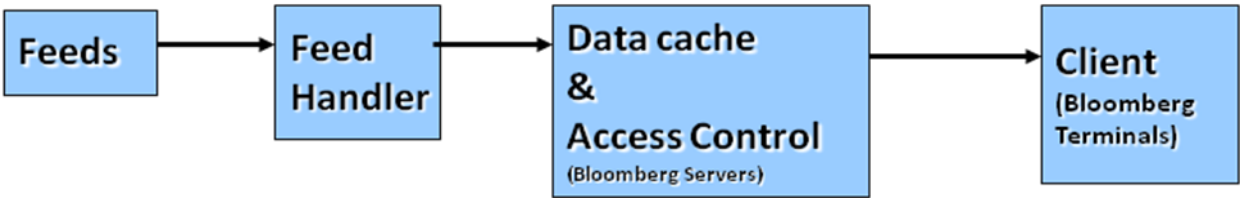
3.4.3. Features

- Clients get direct feeds

- The Feed Handler is separated:
- Line Handler handles networking protocol
- Message Handler normalizes data
- The entire system runs on a single server
- There is no Access Control since the client controls the entire system
- Clients are software applications, does not remote third parties

The Universal Feed Handler Suite is designed to be a highly customizable modular system. The software has support for TCP, UDP and Infiniband so that it can be used in any networking situation. The feed handler is split into two separate components, a message handler and a line handler, that can be reused for different feeds. The line handler handles the incoming network protocol while the message handler interprets and translates the message.

3.5. Bloomberg



[Figure 11: Bloomberg's Architecture]

3.5.1. Market

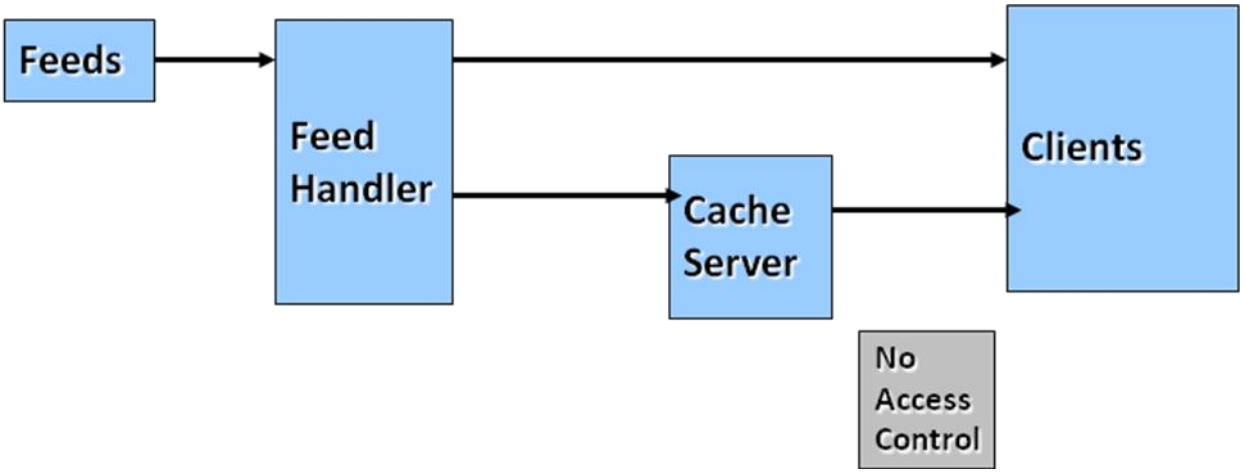
Bloomberg is a world leader in financial news, including television and radio, and data services. The core of Bloomberg is their Bloomberg Professional service, which is delivered to clients via Bloomberg Terminals. The primary function is to provide the client with a wide array of financial market data from sources such as NYSE, NASD and OPRA. However, this service provides a full range of services beyond market data including messaging, historical company information, biographies and more. At over $1500 per month per subscription, the Bloomberg Terminal represents the high end of the market data spectrum. With over 100 thousand users in North America and over 130 thousand worldwide, the Bloomberg Terminal is also one of the most prevalent tools used by hedge funds, insurance companies, banks and other large financial institutions.

3.5.2. Features

- Bloomberg Terminals are dumb terminals
- Bloomberg Server is a big black box that handles all sorts of real-time communications

Bloomberg Terminal is only a dumb terminal; it connects to Bloomberg servers over a dedicated network. This massive private network is one of Bloomberg's primary assets, allowing them to bypass congested public networks. Bloomberg has also recently started to offer another option for financial market data – Bloomberg B-Pipe. B-Pipe is simply an aggregated data feed from Bloomberg without all of the extras that come with a Bloomberg Terminal. There is no feed, just a platform independent stream of data.

3.6. Real-Time Innovations (RTI)



[Figure 12: Real-Time Innovations' Architecture]

3.6.1. Performance Number

Latency under 65 microseconds

3M messages per second

3.6.2. Features

Feed Handler directly sends to independent applications

Multicast Communication

3.6.2.1. Independent Applications

Messages are sent directly from feed handlers to subscribing applications. They call this peer-to-peer architecture. Every application is independent. No resources are shared between different operating system processes. This helps minimize latency.

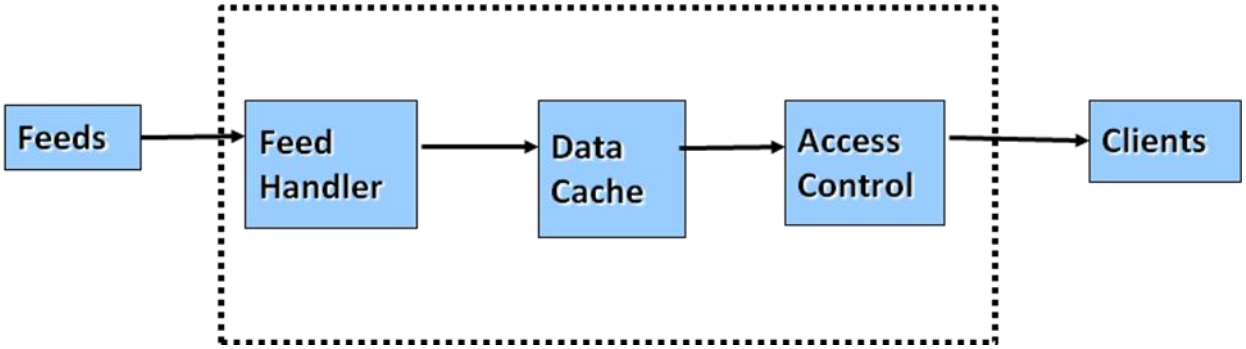
3.6.2.2. Multicast Communication

Multicast communication minimizes latency and maximizes throughput. It allows messages to be delivered to all subscribers simultaneously. The switch only delivers the message to the nodes, which there is a subscriber, with no processing time imposed on other processors or network ports.

3.6.2.3. Channel Properties

Processes communicate with each other over channels. Each channel has its own properties, allowing the messages to be tuned. They call this Quality of Service Tuning. RTI allows the same message to be published over multiple channels. Each subscriber then selects the appropriate channel with QoS requirements.

3.7. Exegy



[Figure 13: Exegy's Architecture]

3.7.1. Performance Number

Exegy Ticker Plant / Infiniband

- 1.68 million updates per second inbound (3.4 million updates per second aggregate, across redundant lines) while sending every update to two consumers
- A single server could handle 2.4 times OPRA's projected rate for January 2008
- Mean latency of just 80 microseconds at 1 million updates per second (2 million updates per second

aggregate inbound across redundant channels)

- 99th percentile latency of just 150 microseconds from 5 Kups to 1.0 Mups

3.7.2. Market

Initially 21 Wall Street customers on March 2007.

3.7.3. Features

- Hardware Acceleration
- Uses FPGA for fast execution speed
- Reconfigurable (Redesigning) Hardware
- Xilinx Virtex-II FPGAs. High performance, high density, and available software development tools

3.7.3.1. Parallel Processing

It helps handle multiple data. By processing in parallel, one instruction per cycle can be performed in instructionless manner. (Instruction means a single operation of a processor.)

3.7.3.2. Approach with FPGA

- Exegy has combined software with reconfigurable hardware to deliver applications that perform at hardware processing speeds, while retaining the flexibility of software.
- Adaptable application logic requiring high data throughput and fast execution speeds is delivered with reconfigurable hardware.
- Xilinx® Virtex™-II FPGA, provides a high degree of functional flexibility.
- Software is also used to supplement the reconfigurable hardware functionality for parts of the application that do not require high data throughput or high execution speeds.
- Using Virtex-II FPGAs, this approach delivers an extremely high throughput application, sustaining a rate as fast as 5Gbps per appliance in Exegy's production version and more than 8Gbps sustained in the next generation of appliances. This represents, in many cases, hundreds of times more throughput than traditional approaches. Additional benefits include virtually zero latency and the ability to add appliances for near linear throughput gains.

Exegy Ticker Plant consolidates exchange market data into a virtual order book allowing customers to

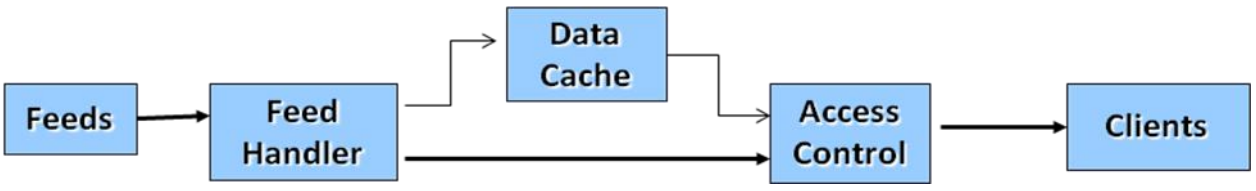
focus on all relevant orders side by side or arrange the orders into a price-aggregated view. Exegy appliances search, transform, filter and analyze massive data stores and high-volume real-time data feeds at high speeds.

3.8. Categorization

Company	Areas of Optimization			Type		Application
	Network	S/W	H/W	Consolidator	Direct Feed Handler	
TA				O		
Reuters	O			O		O
Wombat		O			O	
Bloomberg	O			O		O
RTI	O			O		
Exegy			O		O	

[Figure 14: Categorization of Competitors Architecture based on their optimizations]

3.9. Ideal Architecture



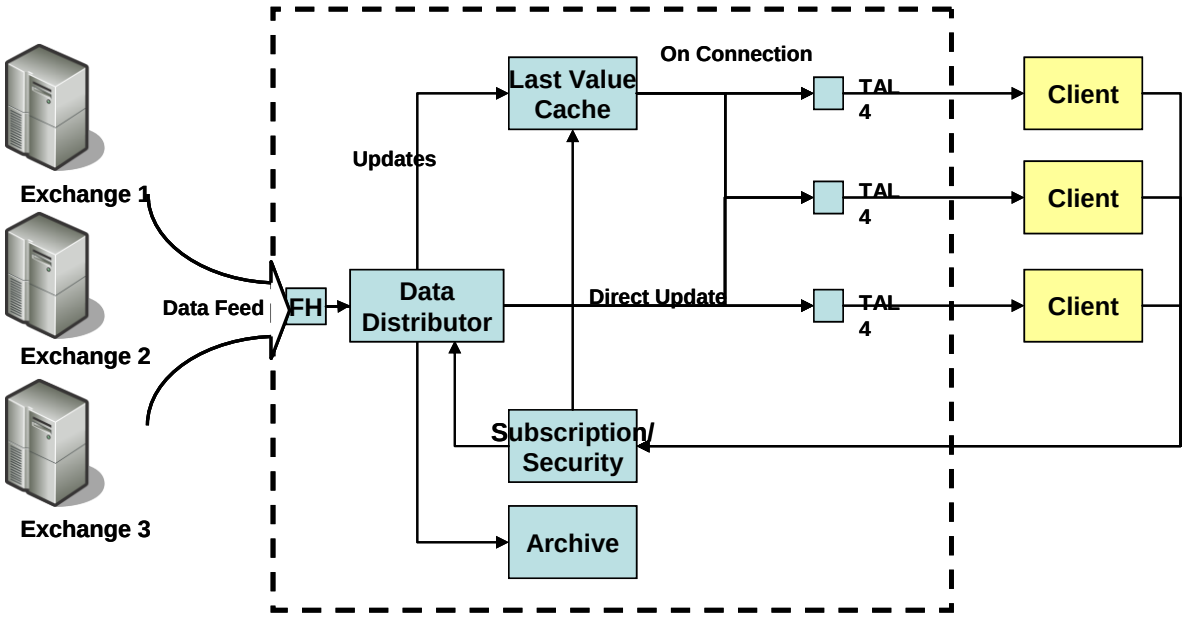
[Figure 15: Ideal Architecture]

Our ideal architecture is motivated by having the shortest path from the feeds to the client. The difference between our ideal architecture and generic architecture is that there is a direct connection between the Feed Handler and the Access Control. The Feed Handler will send all ticks to both the Access Control and the Data Cache. The Data Cache will update itself appropriately and the Access Control will send the updates to the appropriate clients. This means that updates that are partial records will be propagated to the clients without filling in the missing data from the Data Cache. This has two effects that can potentially speed up the process. The first is that the messages will simply be smaller, requiring less bandwidth. The second is that the missing data will not be filled in to create a complete

record, thus bypassing the Data Cache en route to the client. The potential downside to this is that it will place the burden of applying the updates on the client-side software which may be more cumbersome than simply replacing old records with new records.

Our system would be connected with the fastest networking technology available and feasible. Currently Infiniband seems to be the fastest choice, although this is highly susceptible to change in the near future as faster versions of Ethernet are created. We could also potentially use FPGAs to speed up the Feed Handler.

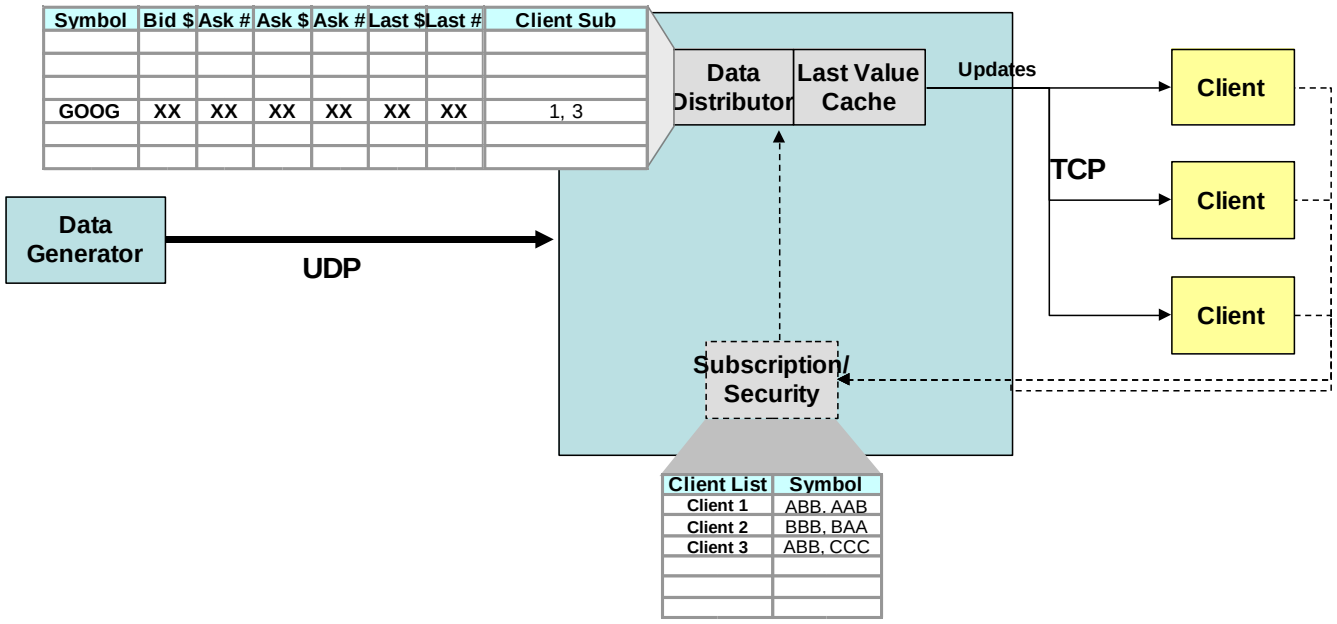
4. Detail Design



[Figure 16: Detail Ideal Architecture Design]

4.1. Overview

Based on the research conducted on the competing architecture of the market leading companies and the ideal architecture, detail design for the project has been defined. The exchanges send data to the feed handler, which then translates the various data protocols into an internal data format. The data is then sent to the data distributor, which will distribute the data to the clients, last value cache and archive. It will determine who to send the data to base on information received from the subscription/security component. The last value cache will store the most recent values received for each symbol. When clients log in the last value cache sends them all relevant data based on their subscription. For the rest of a client's session, data is received directly from the data distributor to fill in the updated data fields as the feed handler receives them.



[Figure 17: Current Development Status of Ticker Plant System]

5. Prototype Design

The prototype design was slightly modified from the proposed design in order to accommodate for the period of a single semester. We built four components: the data generator, clients, subscription/security, and a single component encompassing the work of the data distributor and last value cache. In our research and testing, we focused on ways to improve the last value cache's performance, leaving optimization of the other components as future work.

5.1. Data Generator

Data Generator is a dummy component to simulate the data feed from the exchange. The main purpose of this component is to manipulate the messages so that it allows having control on the data that is generated which will eliminate any uncertainty coming from various data size. It is designed to control the length of the character per symbol and the number of messages that will be generated. It will allow to have some prediction on the behavior of the performance measurement.

The structure of the data generated will contain fields as the following.
<Symbol><Ask Price><Ask Quantity><Bid Price><Bid Quantity><Last Price><Last Quantity>

The size of the data generated is 57 bytes in fixed size containing random number in each of the field.

The size of the symbol can be modified by adjusting the length of the symbol (e.g., length of symbol 3 would produce ABC, APL, FTI, etc . . .) and adjusting the size of the character set used(e.g., {A-Z}, {A-G}, {A-M}). This allows modification of both symbol length and the number of unique symbols.

5.2. Data Distributor/Last Value Cache

The last value cache works by storing the data it receives from the data generator in a hash table (the core of the software group's research was optimizing the performance of a hash table). The key for the hash table was the symbol while the value was the ticker data as well as a client listing. This client listing represented the clients subscribing to a symbol. This allowed the last value cache to act as the data distributor, automatically sending updates to the clients by retrieving the subscription listing alongside the ticker data.

In the completed implementation, the last value cache will be separated from the data distributor in order to off-load the work from one component into two separate components.

5.3. Client

The client receives the data from the data distributor/last value cache. It can update its subscription settings by contacting the subscription/security component.

5.4. Subscription/Security

The subscription/security component is designed consisting a hash table with clients as the keys of the table and their subscription lists as the value. When a client logs on the subscription list is retrieved and sent to the data distributor/last value cache which then handles the sending of data. If a client wishes to change subscriptions, they must contact this component.

5.5. Network Connection

Data generator will be connected to the Last Value Cache using Ethernet connection. Data Generator will be using UDP and Client will be using TCP to connect with the LVC. The main reason for using UDP to connect Data Generator to the LVC is because we don’t have to secure the incoming message from getting lost through the network.

6. Optimization

6.1. Hashing Algorithm

A hash table is a data structure that associates keys with values. The hash table is primarily used for efficient lookup: that is, given the key, i.e. Person's name, obtain the corresponding value, i.e. Person's address. The hash table is composed of two parts: a container, the actual table where the data is stored, and a hash function, which maps the hash generated to an address in the container. The hashing table can be optimized by improving the container's and the hash function's algorithm. Townsend Analytics is using C++ STL for their hash container and CRC32 for their hash functions. The goal for the optimization is to show that there are other implementations of the hashing container and hash functions that can improve the overall of the hashing algorithm.

Three hash container algorithms were investigated in the report. The first container algorithm is the standard hash map provided by the C++ Standard Library. In this algorithm, the data is stored in an array of linked list. The second container algorithm is the sparse hash map. The sparse hash map stores the data in a Sparse table container, which is an array that uses very little memory to store unassigned indices, using internal quadratic probing. The third container algorithm is a Dense hash map. Similar to sparse hash map, it uses internal quadratic probing to store the data into the hash table. However, instead of using some Sparse table, it uses a C array as the container, to eliminate the memory management overhead that the Sparse table contains. In essence, the sparse hash promises memory efficiency in the expense of speed, the Dense hash promises speed in the expense of memory. Below is the performance numbers for the three hash container algorithms using a machine with a Pentium 4 2GHz and 2G of memory. The Dense hash map shows an improvement of almost twice as the standard hash map.

[Figure 18: Benchmark Result on Dense and Sparse Hashing Algorithms]

SPARSE_HASH_MAP:
map_grow 665 ns
map_predict/grow 303 ns
map_replace 177 ns
map_fetch 117 ns
map_remove 192 ns
memory used in map_grow 84.3956 Mbytes

DENSE_HASH_MAP:
map_grow 84 ns
map_predict/grow 22 ns
map_replace 18 ns
map_fetch 13 ns
map_remove 23 ns
memory used in map_grow 256.0000 Mbytes

STANDARD HASH_MAP:
map_grow 162 ns
map_predict/grow 107 ns
map_replace 44 ns
map_fetch 22 ns
map_remove 124 ns
memory used in map_grow 204.1643 Mbytes

Source: http://google-sparsehash.googlecode.com/svn/trunk/doc/performance.html[27]

The hash function is also critical in determining the overall speed of the hash table algorithm. There are a couple of open source hash functions that maybe used to improve the CRC32 algorithm that Townsend

Analytics is using. As summarized by the table below, the test is done using an AMD athlon XP 1.620 Ghz. Each hash function was tested on how many seconds it took to hash a random buffer of 256 bytes 5 million times. According to the benchmark displayed below, four hashing functions performed better than CRC32. One of these hash function is the SuperFastHash, which performed at least 3 times faster than CRC 32.

[Figure 19: Benchmark Result on Different Hash Function]

	AMD Athlon XP 1.620Ghz				Power4 1Ghz	UltraSparc III 1.2Ghz
	Intel C/C++ /O2 /G6 /Qaxi /Qxi /Qip	**MSVC** /O2 /Ot /Og /G6	**WATCOM C/C++** /otexan /6r	**GCC** -O3 -march=athlon-xp	**GCC** -O3 -mpowerpc64	**CC 32bit** -O
CRC32	6.42	5.66	5.66	5.67	14.06	8.75
One at a Time	5.76	5.66	5.66	5.69	12.79	5.57
Alpha Numeric	3.29	4.06	4.06	5.67	10.26	5.52
FNV Hash	4.88	4.84	4.83	4.87	8.92	11.98
Bob Jenkins	2.08	2.36	2.03	2.07	6.16	3.08
SuperFastHash	1.54	1.92	1.59	1.34	3.71	2.15

Data is time in seconds taken to hash a random buffer of 256 bytes 5 million times. Download test here

Source: http://www.azillionmonkeys.com/qed/hash.html[28]

6.2. Multi Threading / Multi Core Processor

Threads are a way for a program to fork itself into two or more simultaneously running tasks. A multi-core CPU combines two or more independent cores into a single package composed of a single integrated circuit. The fundamental goal of multithreading and multiprocessing is the improvement of performance through additional processors. This performance improvement is measured in terms of “speedup”, which relates changes in performance on a workload to the number of CPUs and threads available to perform the work. We have implemented Last Value Cache using multi thread to input output and hash the data. If we create multi threads and use multi process in order to distribute some tasks of each component, we would expect the effect in terms of the speed in our system.

7. Experimental Results

7.1. Objective

Main objective of the experiment is to provide a benchmark through the actual testing based on the prototype that has been built. It is to see how each the program behaves relative to different factors. It will provide future goals on how to optimize and develop the system.

7.2. Assumption

Following were some of the assumption for the experimental approach.

- *'Performance will be improved by using higher performance equipments'.* Since the experimental tools used for the unit testing and the system testing is based on the off-the-shelf hardware, it will give greater improvement using high-performance equipment that will be used in the actual practice.
- *'The relative performance rate will be consistent regardless of what machine it will be tested on'.* In order words, it means that the absolute performance may differ, but the overall result should be same. For example, if there are two different algorithms tested on PC1 using two algorithms A and B, and if it showed that A is faster than B, then it should work same with any other PCs showing the same result. The latency may differ due to the different PC specification, but the result saying that A should be faster than B should always be the same.
- *'Percentage of improvement is more important than the absolute measurement.'* Since testing result is based on the simulation of using dummy data, it will not indicate the actual performance. However, it will give a good indication of how much it can be improved potentially using such optimal algorithms.
- *'Numbers of data that will be generated can be manipulated by the data generator to discover the number of data that was received and lost through the transactions.'* Since UDP is used for transmission through the network, it is important to know how many data is generated and how many data is actually received. The prototype system is developed to allow the have precise measurements of each components.
- *'For testing the hashing algorithm, performance will be measured from the point of actual updates made to the hash table and discard any measurements for inserting data.'* Based on the research, LVC will be fed with data constantly and the entire hash table will be already filled up containing symbols in each fields. Therefore, the caching data will only require updating into the memory.

7.3. Unit Testing on LVC

7.3.1. Description

The main purpose of the unit testing was to test the performance of each hashing algorithm and their behavior depending on the variables such as length of characters and the table size determined by the unique symbols generated. It will give precise measure on the component eliminating any networking factors.

7.3.2. Testing Specifications

The test was performed varying the number of unique symbols and the length of the character. The length of character and the variation of different alphabets used in each character field determine number of unique symbols. Calculating the number of unique symbol is derived by using the following formula:

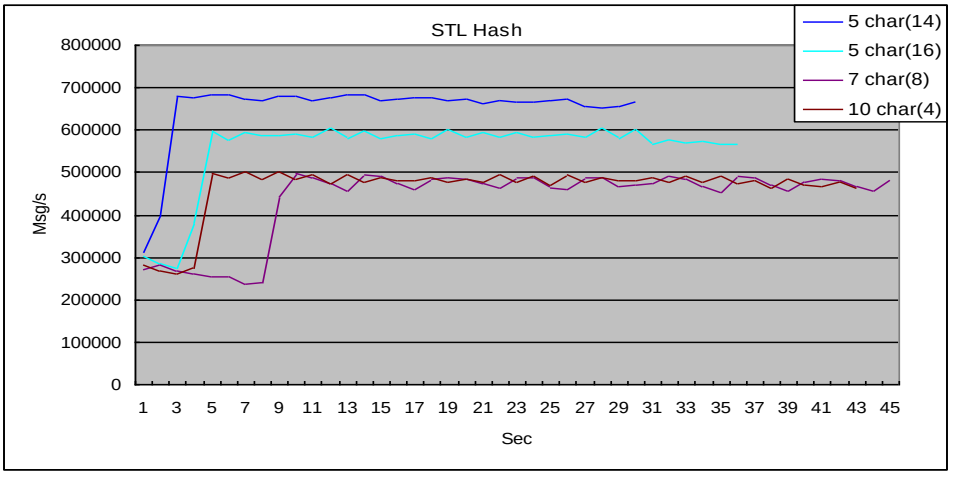
(Unique Characters) ^ (Length of Character)

For the length of character, 5, 7, and 10 characters and 0.5M, 1.0M and 2.0M unique symbol is used for the testing. Having the variation of length and unique symbols, and random number in other fields is generated at the size of 57bytes and pushed into different hashing algorithm.

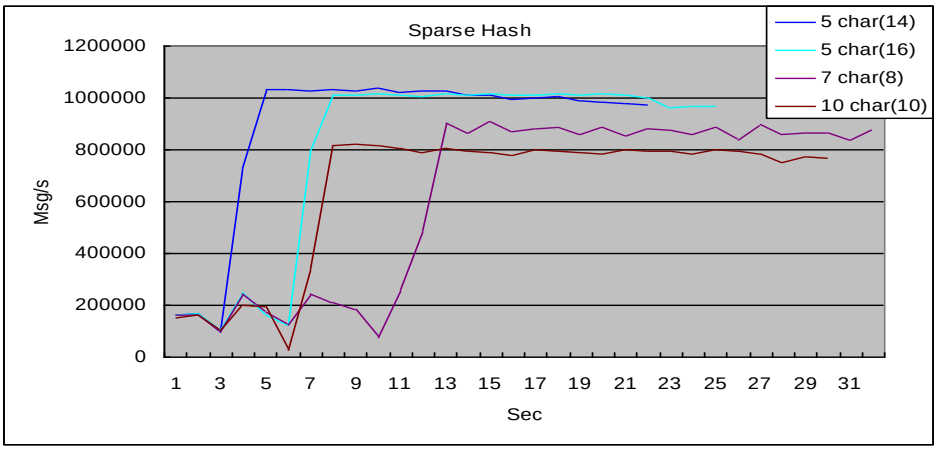
In addition, 20 million messages is generated and hashed into the each of algorithm to observe the performance of inserting and updating the data into the hash table.

Machine
Intel(R) Core(TM)2 CPU T7200 @ 2.00GHz x86 Family 6 Model 15 Stepping 6
Cache Size 4096 Kb
2.00 GB of RAM
Operating System – Windows XP SP2

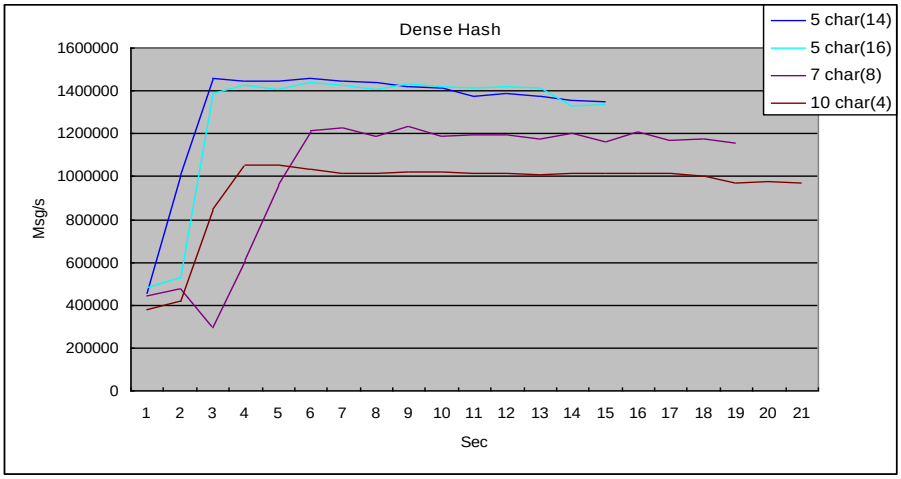
7.3.3. Testing Results



[Figure 20: Benchmark Result on Unit testing using STL hash]



[Figure 21: Benchmark Result on Unit testing using Sparse hash]



[Figure 22: Benchmark Result on Unit testing using Dense hash]

Number of Characters	Unique symbols	Memory used	Performance (Msg/s)
5 char	537,824 (=14^5)	67 Mb	671,600
5 char	1,048,576 (=16^5)	129 Mb	585,771
7 char	2,097,152 (=8^7)	257 Mb	475,649
10 char	1,048,576 (=4^10)	129 Mb	482,512

Performance of STL is impacted by both character length and the unique number of symbols

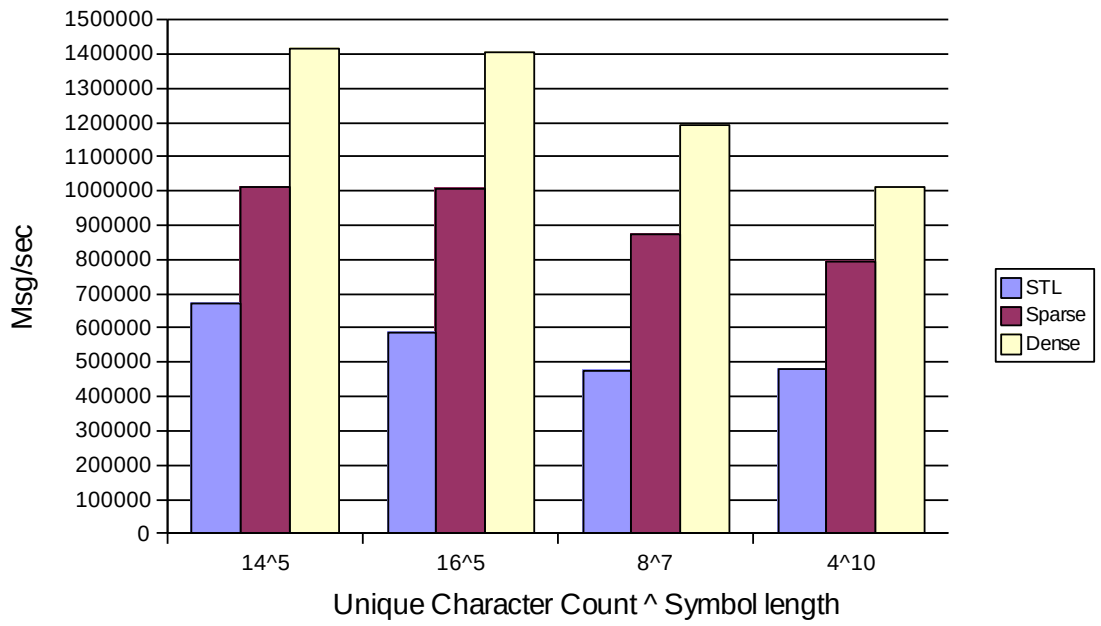
Number of Characters	Unique symbols	Memory used	Performance (Msg/s)
5 char	537,824 (=14^5)	56 Mb	1,012,201
5 char	1,048,576 (=16^5)	110 Mb	1,003,827
7 char	2,097,152 (=8^7)	218 Mb	872,994
10 char	1,048,576 (=4^10)	109 Mb	793,005

Performance of Sparse hash is impacted by character length but not by the unique messages which determines the table sizes

Number of Characters	Unique symbols	Memory used	Performance (Msg/s)
5 char	537,824 (=14^5)	101 Mb	1,412,862
5 char	1,048,576 (=16^5)	136 Mb	1,404,268
7 char	2,097,152 (=8^7)	273 Mb	1,192,174
10 char	1,048,576 (=4^10)	137 Mb	1,013,039

Performance of Dense hash is impacted by character length but not by the unique messages which determines the table sizes

[Figure 23: Overall Performance Comparison on the Hashing Algorithms]



7.3.4. Analysis

As shown by the test results on the unit testing, there was a remarkable improvement on performance using dense hashing algorithm. It also showed that Sparse and Dense Hashing algorithm did not rely on the table size. Rather, it was determined by the length of the character used for the hashing. It seems that since Sparse and Dense is optimizing hash algorithm on the container part, it shows that even the table size increased it should no significant changes on the performance.

7.4. System Testing

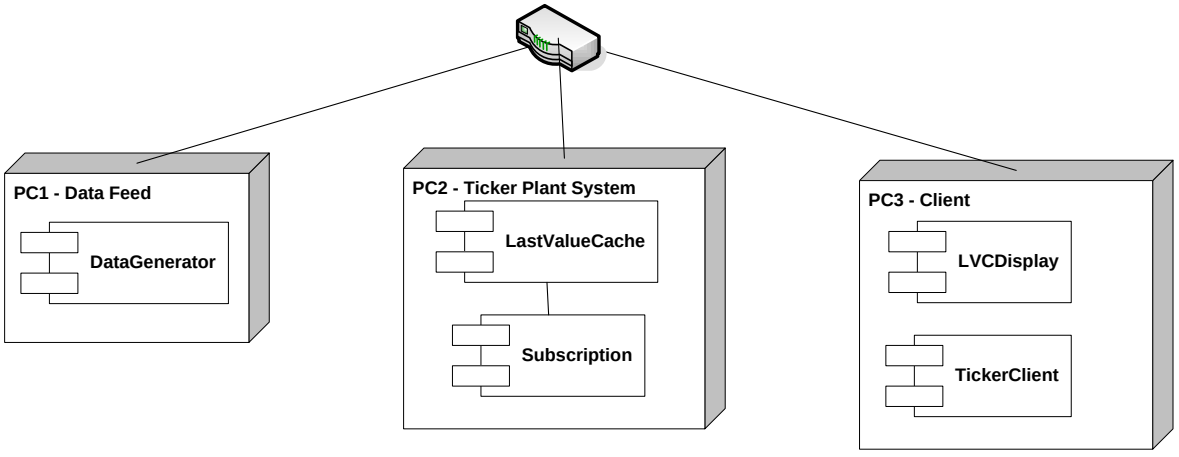
7.4.1. Description

The main goal of system testing is to connect all the components and measure the overall performance of the system. With all the components connected together, the data can be streamed into the last value cache to measure how much throughput it can handle. In addition to that, possible bottlenecks other than the last value cache, such as the limited network bandwidth, can be identified.

7.4.2. Testing Specifications

The test was performed using three systems, which all have the same specifications. Each of these machines were running Intel Core 2 CPU @ 2.40 GHz and were connected using Belkin Pre-N Router with a maximum throughput of 108 Mbps. The first system contained the Data Generator component, which created random ticker data. The second system contained the Last Value Cache and Subscription components, which stored the ticker data in a hash table and sent the ticker data to subscribed clients. The third system contained the LVC Display component, which controlled the Last Value Cache and displayed its performance in a graph, and the Client component which received the ticker data it subscribed to in the last value cache.

Machines 1 - 3
Intel® Core 2 CPU @ 2.40 GHz
2.00 GB of RAM
Operating System – Windows XP SP2
Broadcom NetXtreme 57xx Gigabit Controller
Network Equipments
Belkin F5D8230-4 Wireless 802.11x Pre-N Router (Maximum throughput of 108 Mbps)
UTP Cable 100 baseT



[Figure 24: Current Architecture of Ticker System at Townsend Analytics]

7.4.3. Testing Results

There were three test sets that were performed during the system testing. For each test set, three hashing algorithms were investigated—STL, Sparse and Dense. For the first test set, one Data Generator was used to feed data into the Last Value Cache. The Data Generator supplied 82,000 messages per second and 82,000 messages per second were handled by LVC. There were no differences in the performance of the three hashing algorithms. For the second test set, two Data Generators was used to feed data into the Last Value Cache. The two Data Generators supplied a combined total of 120,000 messages per second while the last value cache was able to handle 109,000 messages per second. There were also no differences in the performance of the three hashing algorithms. For the third and last test set, three data generators were used to feed data in the last value cache. Interestingly, the third set produced the same results as the second set.

- **One Data Generator**

Hash Algorithm	Data Generator (msg/sec)	Last Value Cache
STL	82,000	82,000
Sparse	82,000	82,000
Dense	82,000	82,000

- **Two Data Generators**

Hash Algorithm	Data Generator (msg/sec)	Last Value Cache
STL	120,000	109,000
Sparse	120,000	109,000
Dense	120,000	109,000

- **Three Data Generators**

Hash Algorithm	Data Generator (msg/sec)	Last Value Cache
STL	120,000	109,000
Sparse	120,000	109,000
Dense	120,000	109,000

7.4.4. Analysis

As shown by the test results of the system testing, the last value cache was not the bottleneck of the system. Instead, the limited bandwidth of the networking equipment used was the major hurdle in the

performance. In this case, the router only had a maximum throughput of 100 Mbps. The total size of one message that is transmitted through the network is 85 bytes (57 bytes for the payload and 28 bytes for the packet overhead). Having said that, the router used in the system testing can only handle a maximum of 120,000 messages per second. There is a need to find ways to get around this bandwidth limitation by using more advanced networking technologies and reducing the percentage of the overhead in the total message size.

8. Conclusions

Based upon the hardware and software analysis and tests, it was determined that hashing algorithms have an affect on the latency of the system. In the hash algorithm tests against the benchmark STL, both the sparse and dense hashing algorithms outperformed the benchmark. Through these tests, it was also determined that the RAM had no effect on the results and was not a factor in latency or throughput. Additionally, performance gains could be seen on the proposed architecture and could be optimized for further performance and identification of bottlenecks in the system.

9. Future works

The base system that was developed during the current term can be tested and modified to optimize the source code. Additionally, the proposed system can move into a prototype phase and be constructed to analyze its performance. Once a stable system is in place, it can be tested on different hardware configurations to identify bottleneck and the optimal system. Further research into exotic technologies such as Field-Programmable Gate Arrays (FPGA) and cell processors, such as those used in the Sony Playstation 3, would be explored in the area of parallel computing. Other hardware research will be conducted in the area of multi-core processing and high-performance computing, as well as 64-bit versus 32-bit computing. Later IPRO's will have the opportunity to test market data system on these platforms and further identify ways to optimize the hardware and software.

10. Obstacles

Obstacles that were faced by the team mainly had to deal with technical issues. The two constraints on the technical side dealt with hardware and networking constraints. On the human resources side, the lack of programmers and programming experience became a roadblock, which cost time. The final constraint the project faced was time, which was a commodity that was difficult to maintain in the completion of project goals.

A. Appendix I

A.1. Result from extracting OPRA Data (FAST)

A.1.1. OPRA FAST Translator

A FAST translator was written to translate sample OPRA data we received from Townsend Analytics. The translation of individual FAST messages is handled by a combination of code downloaded off the OPRA website (http://www.opradata.com/specs/data_recip.jsp) and the FAST API(http://www.fixprotocol.org/fast). In addition to this, code needed to be written to parse out the individual FAST messages from the data. The steps involved in this are detailed further in this report.

The translator currently translates the messages, however it does not do anything with them after that other than printing out the data for one type of message(the category K messages). The hope is that the code should be easily extended to be able to extract the data our system requires and send this to the Last Value Cache component, effectively replacing the current random data generator.

A.1.2. Parsing out the FAST messages

The data is partitioned into frames (which, to my understanding, represent on packet of data). Each frame contains several FAST messages.

The first four bytes of a frame include some flags (Townsend did not give any further details as to what these flags represent) followed by the frame size in the 16 least significant bits. The next 4 bytes specify the OPRA UDP channel being used (a value between 0 and 24). Assuming S to be the frame size specified above, the next S bytes will contain the actual frame of data. The first byte in the frame should always be 0x01 to specify the Start of Header and end with 0x03, the End of Transmission. The byte following the 0x01 will specify the size of the message (limiting messages to the size of 0xFF, or 255 bytes). The following bytes (until the message size is reached) will be FAST data followed by another message size byte and message. This continues until the end of frame at 0x03.

B. Appendix II

B.1. User manual

B.1.1. Introduction

The Ticker Plant System is designed to test the performance of the system. There are five executable files; Datagenerator.exe, LVC.exe, LVCDispaly.exe, Client.exe and LVC-unit.exe. You can experiment with parameter settings on your own to have better understand this tool.

This documentation will outline what you need to run Ticker Plant System, the tool's important features, and the steps required to perform a test.

Remember that this is a beta, or preliminary, version of this application, which means that it is still in development for next IPRO team.

B.1.2. Ticker Plant System Tool Overview and System Requirements

You may download the Ticker Plant System Tool to run on a local machine as a stand-alone application or you may run it on network as a server and clients. At minimum, you will need:

Operating system: Windows 95/98/NT/XP/VISTA

- *If having trouble executing LVC, LVC-unit; need to compile the program with C++ compiler*
- *If having trouble executing Client, LVC*-Display; need to install Java software. Follow the below instructions

Java software: Java Runtime Environment/Java Plug-In version 1.6

- JAVA can be downloaded through the following website:

http://faculty.ed.umuc.edu/~arnoldyl/NetBeansTutorials/Install-Jdk.html

Network connection: In addition to requiring a network connection to use Ticker Plant System online, you will also require one to generate the data and the other to subscribe the data as clients.

B.1.3. Workflow Overview

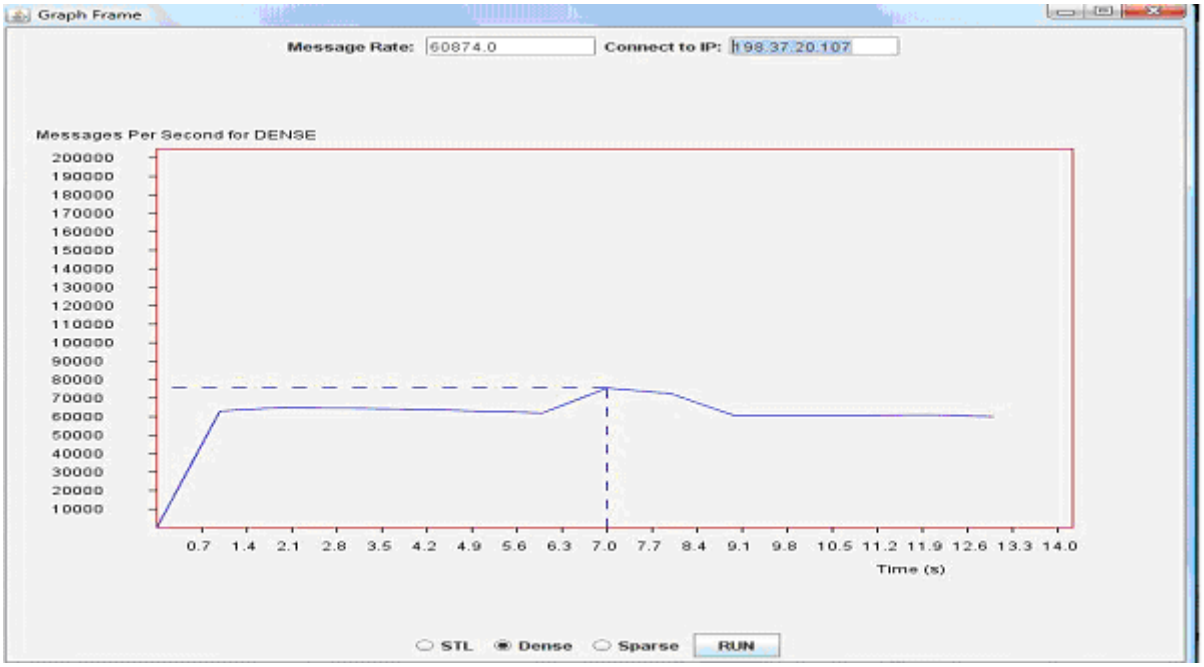
The steps below serve as a general guide for the sequence you will follow to test this tool.

B.1.3.1. Run LVC.exe

LVC screen will show and ask whether you want to use the network to connect. It will show you the current state of the connection (UDP/TCP). It will then wait for the datagenerator to send the message

B.1.3.2. Run LVCDisplay.exe

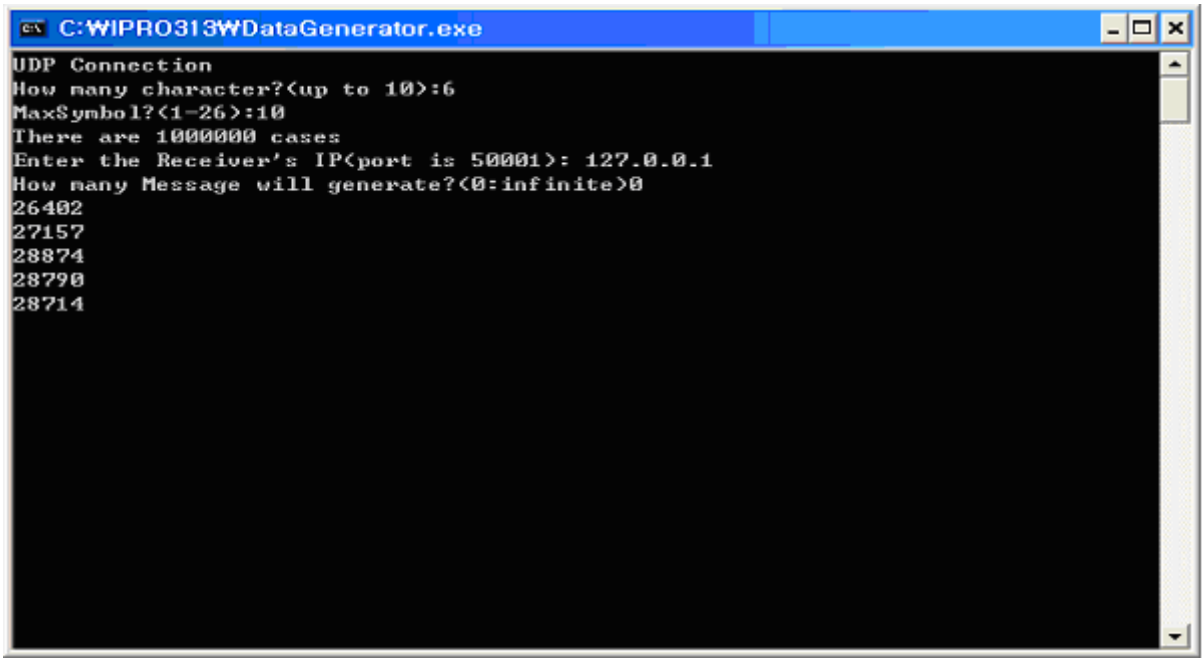
LVCDisplay screen shows the messages per second for the hashing algorithms.



- Choose one of the hashing algorithms we want to test.
- Input IP Address

B.1.3.3. Run DataGenerator.exe

After you have setup the LVC (Last Value Cache), run DataGenerator.exe file to send messages. If you have run the file, you will see the protocol on the first line (UDP/TCP).



- *'How many character?<up to 10>:'*

Input the length of symbol characters.

- *'MaxSymbol?<1-26>:'*

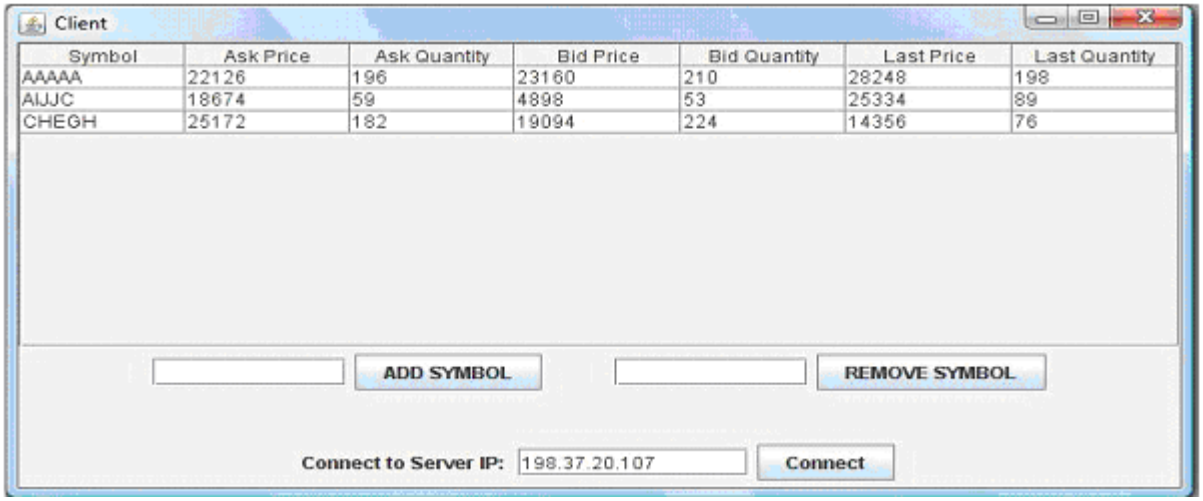
Input the number of characters for each symbol label.

- Input '127.0.0.1' as an IP address if the program is executed in a local machine. Otherwise, type in the IP address of the Machine that contains the LVC.exe program..

- *'How many Message will generate?<0:infinite>*

  Input the number of message you want to generate for the test.

B.1.3.4. Run Client.exe

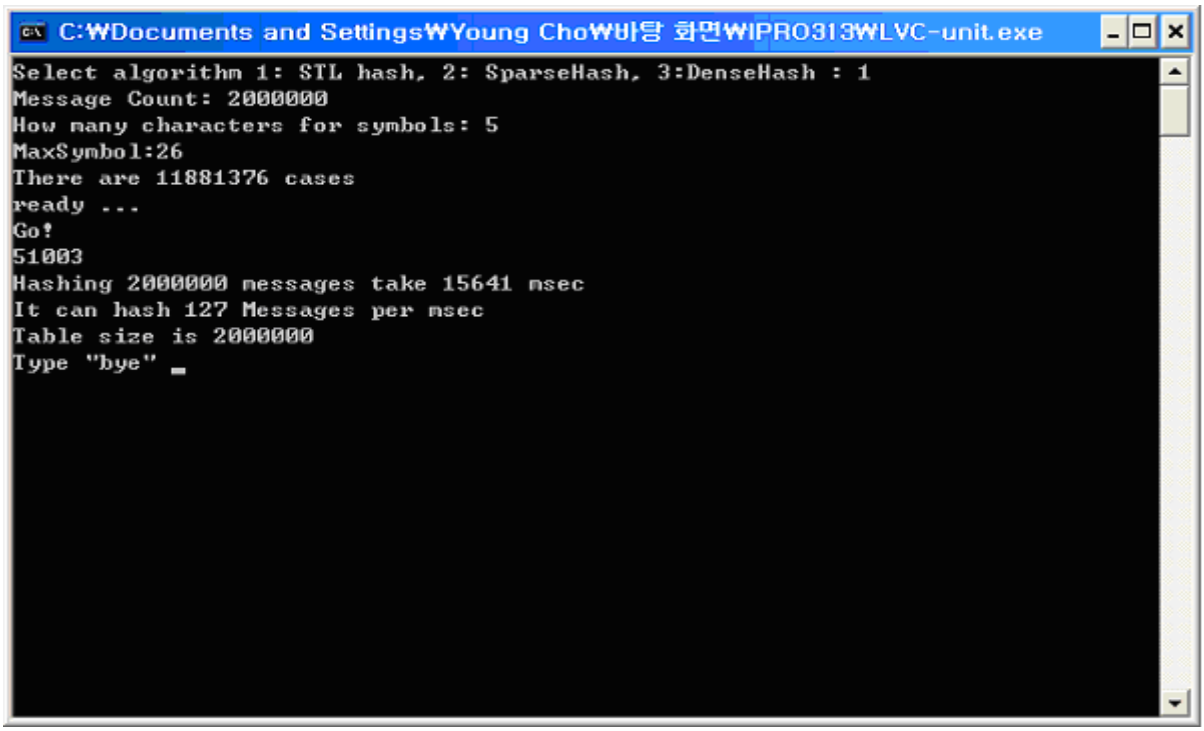


- Input Data Generator's IP Address.

- You may want to add or remove symbols by typing it.

B.1.3.5. LVC-unit.exe

LVC-unit is run in a same manners as Data Generator, except the fact that it will be run as an independent component to execute both data generation and caching data into hash table.



- *'Select algorithm 1: STL hash, 2: SparseHash, 3:DenseHash:'*

Choose one of the hashing algorithms we want to test.

- *'Message Count:'*

  Input the number of messages you want to generate

- *'How many characters for symbols:'*

Input the length of symbols.

- *'MaxSymbol:'*

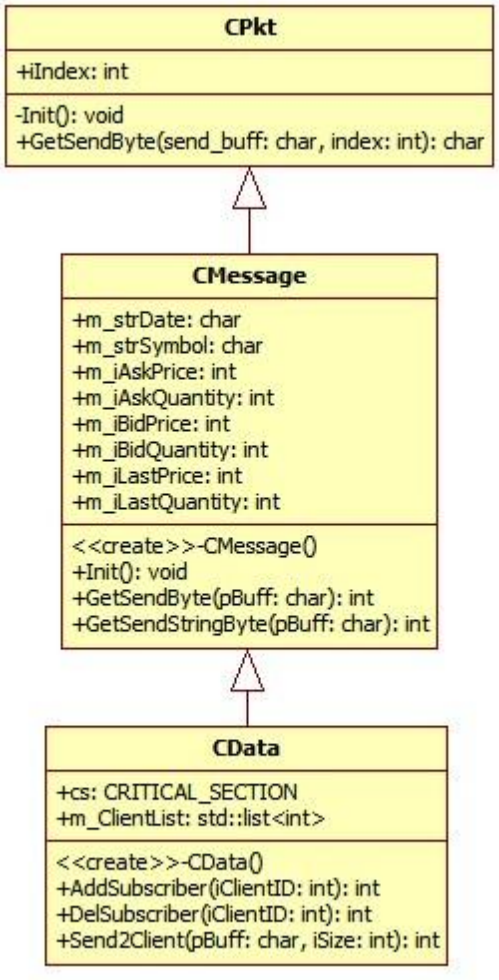
Input the number of characters for symbol

B.2. Technical Manual

B.2.1. Development Environment

- **Operating System**: Windows
- **Compiler:** MSVC (Visual Studio 2005)
- **Library**: Winsock2. SparseHash-0.8 *(http://code.google.com/p/google-sparsehash/)*

B.2.2. Message Class



<Figure 2. Message Class Diagram>

CPkt class is the interface of general message.

CMessage class is for packet between the data generator and LVC.

CData is the data structure for the cache. Difference between CMessage and CData is that CData has the list which contains clients who are interested in this message.

```
int CMessage::GetSendByte(char *pBuff)
{
	int iSize = 0;
	SetString(pBuff, m_strDate, DATESIZE, iSize);
	SetString(pBuff, m_strSymbol, SYMBOLSIZE, iSize);
	SetInt(pBuff, m_iAskPrice, iSize);
	SetInt(pBuff, m_iAskQuantity, iSize);
	SetInt(pBuff, m_iBidPrice, iSize);
	SetInt(pBuff, m_iBidQuantity, iSize);
	SetInt(pBuff, m_iLastPrice, iSize);
	SetInt(pBuff, m_iLastQuantity, iSize);

	return iSize;
}

CMessage Message;
Message.m_iAskPrice = 1;   //set values
.
.

index = Message.GetSendByte(pBuf);
//UDP
if(!(iSize = sendto(m_Socket, pBuf, index, 0, (struct sockaddr *)
&m_Address, sizeof(m_Address))))
```

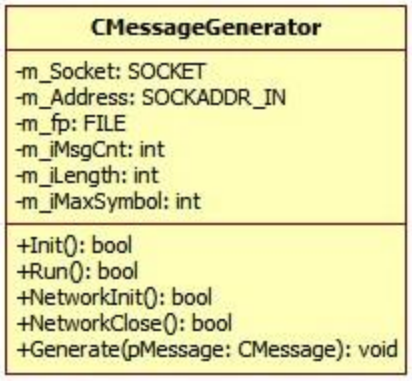
< Example 1. Sending a Message>

```
void DataDa::Parse(char *pBuff, CMessage *pPkt)
{
	int index = 0;
	GetString(pPkt->m_strDate, pBuff, DATESIZE, index);
	GetString(pPkt->m_strSymbol, pBuff, SYMBOLSIZE, index);
	pPkt->m_iAskPrice = GetInt( pBuff, index);
	pPkt->m_iAskQuantity = GetInt( pBuff, index);
	pPkt->m_iBidPrice = GetInt( pBuff, index);
	pPkt->m_iBidQuantity = GetInt( pBuff, index);
	pPkt->m_iLastPrice = GetInt(pBuff, index);
	pPkt->m_iLastQuantity = GetInt( pBuff, index);
}

//UDP
size = recvfrom(hServSock, buff, BUFFSIZE, 0, (struct sockaddr *)
```

```
&servAddr, &servAddrSize);

if(size > 0 && m_iAlgo > 0){
        Parse(buff, &pkt);
}
```

< Example 2. Receiving a message>

B.2.3. Data Generator

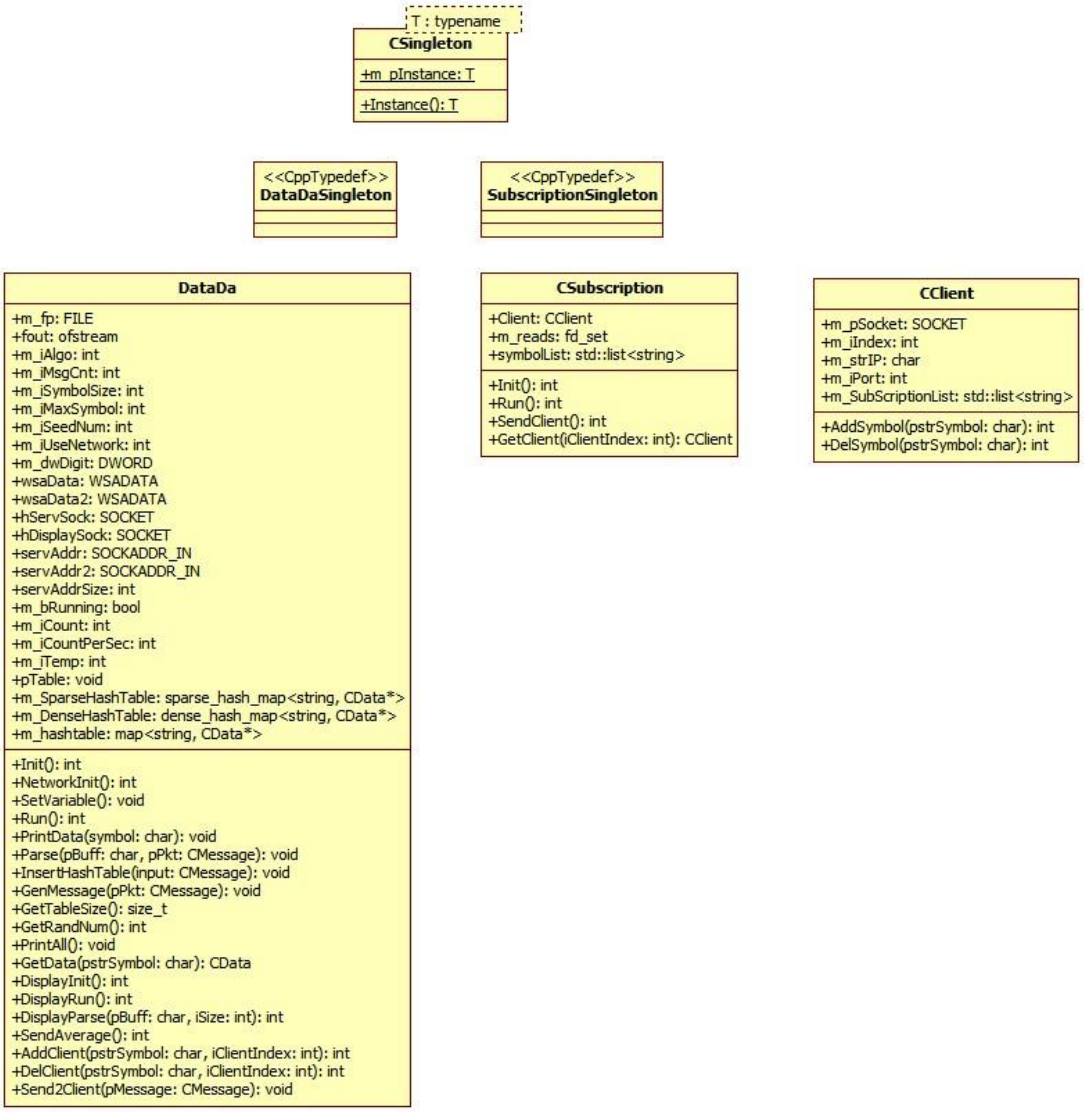


< Figure 3. Message Generator class diagram>

The message generator generates the message. The length of the symbol of the message is the m_iLength variable. Each character has m_iMaxSymbol unique cases.

It generates the symbol in alphabetical order. Others values are generated randomly with 50% probability.

B.2.4. Last Cache Value



< Figure 4. LVC class diagram >

There are 2 singletons for data cache and subscription and 3 threads in the LVC and each singleton runs on different thread. LVC uses different hash table according to the m_iAlgo. (1- Standard map, 2- Sparse hash table, 3- Dense Hash Table). Each record in the hash table has the client list so that it can send to clients without look up another table. In other side, the subscription manages the client and subscription. The subscription has table. Each record has the symbol list and client index as a key to know a client interested in which symbols easily.

C. Appendix III

C.1. References

[1] ***"What is a Stock"***
Douglas Gerlach, 2007 http://www.youngmoney.com/investing/sharebuilder/goals/031021_10

[2] ***"Using Information from Trading in Trading and Portfolio Management: Ten Years Later"***
David J. Leinweber, May 2002

[3] ***"Algorithmic Trading: Its growth and Limitations"***
Bob McDowall, Dec 2005,
http://www.it-analysis.com/business/content.php?cid=8236

[4] ***"Trading At Light Speed: Analyzing Low Latency Data Market Data Infrastructure"***
Johnson Jeromee, Aug 2007

[5] ***"Corporate Overview"***
Townsend Analytics, Sept 2007
http://www.townsendanalytics.com

[6] ***"Ahead of the Tape"***
The Economists, Jun 2007 http://www.economist.com/finance/displaystory.cfm?story_id=9370718

[7] ***"Technology and Exchanges"***
The Economist, Feb 2006 http://www.economist.com/finance/displaystory.cfm?story_id=E1_VQSVPRT

[8] ***"A buy-side handbook Algorithmic Trading"***
The Trade, 2005
http://www.thetradenews.com/files/magazine/algo_1.pdf

[9] ***"Sybase Real-Time Data Services: Information Management Strategies for the Unwired Enterprise"***
Sybase, 2005

[10] ***"Managing Risk in Real-Time Markets"***
Adam Sussman, Feb 2005

[11] ***"The Real Time Data Management Imperative"***
Ken Rugg & Mark Palmer

[12] **Realtick**
http://www.realtick.com/v2_getpage.asp?page=plat
https://www.realtick.com/v2_getpage.asp?page=plat_tald
https://www.realtick.com/v2_getpage.asp?subnav=false&page=plat_tapi
http://208.234.169.12/so/?action=view&SR_ProductID=1

[12] **Taltrade**
http://www.taltrade.com/getpage.asp?page=news_pres&temp=3&ID=845

[13] **RMDS info from Web**
http://about.reuters.com/home/aboutus/professionalproducts/index.aspx
http://about.reuters.com/home/aboutus/ourcompany/keyfacts.aspx

[14] **Securities Technology Analysis Center**
http://www.stacresearch.com/

[15] **Network Speed**
http://www.pixelbeat.org/speeds.html

[14] **Wombat Releases Feed Handler for Bloomberg B-Pipe**
http://www.bobsguide.com/guide/news/12794.html

[15] **About Bloomberg: Bloomberg Professional**
http://about.bloomberg.com/professional/index.html

[16] **Bloomberg B-Pipe Market Data Feed Stirs Up Competition**
http://www.advancedtrading.com/infrastructure/showArticle.jhtml?articleID=196900266

[17] **About Bloomberg: Bloomberg Professional**
http://en.wikipedia.org/wiki/Bloomberg_Terminal

[18] **What is a Bloomberg Terminal?**
http://ids.csom.umn.edu/faculty/kauffman/courses/8420s98/project/bloomberg/abb.htm

[19] **Competition Looms in Bloomberg Real-Time Market Data Feed Use**
http://www.gartner.com/DisplayDocument?doc_cd=138692

[20] **Universal Feed Handler Suite Product Sheet**
http://www.wombatfs.com/products/product_sheets.php?sheet_name=fh

[21] **RTI Data Distribution Service**
http://www.rti.com/products/data_distribution

[22] **RTI White Papers**
http://www.rti.com/resources/whitepapers.html

[23] **The Exegy Ticker Plant 2.0,**
http://www.exegy.com/images/pdfs/XTP2.0web.pdf
http://www.exegy.com/specs.html

[24] **Exegy Performance News**
http://www.ts-a.com/news/news.php?ref=/news/news_nov05.php
http://www.banktech.com/showArticle.jhtml?articleID=202403965

[25**] Infiniband**
http://en.wikipedia.org/wiki/Infiniband_

[26] **FPGA**
http://en.wikipedia.org/wiki/fpga_

[27] **Google Sparse and Dense Hashing Algoritms**
http://google-sparsehash.googlecode.com/svn/trunk/doc/performance.html

[28] **Benchmark on Hash Functions**
http://www.azillionmonkeys.com/qed/hash.html

D. Appendix IV

D.1. Table of Contents for the IPRO Final CD

- **Directory of D:**
- **Directory of D:\Abstract**
  - Abstract.doc
- **Directory of D:\Contact List**
  - IPRO313_Contact List_20070828_v0.2.xls
- **Directory of D:\Final Presentation**
  - IPRO313_Final Report_20071130_v1.0.doc
  - IPRO313_Final IPRO Presentation_20071130_v1.0.ppt
  - IPRO313_TALI Presentation_20071127_v1.0.ppt
  - poster.jpg
- **Directory of D:\Final Report**
  - IPRO313_IPRO Final Report_20071139_v1.0.doc
- **Directory of D:\Meeting Minutes**
  - Meetin_Minutes.doc
- **Directory of D:\Midterm**
  - IPRO-313_Midterm_Report_.doc
  - IPRO 313ETHICS.doc
  - IPRO_313v3.ppt
- **Directory of D:\Project Plan**
  - HW_Project_Plan.doc
  - IPRO313_Software_Group_Project_Plan_20070925.doc
  - IPRO_313_-_Project_Plan_v3.doc
- **Directory of D:\Source Code**
- **Directory of D:\Source Code\0. Executable Files**
  - DataGenerator.exe
  - LVC.exe
  - LVC_Unit Testing Version.exe
- **Directory of D:\Source Code\1. Data Generator_Last Value Cache_C++**
  - IPRO_313.zip
- **Directory of D:\Source Code\2. Client_LVC Display Controller_JAVA**
  - CLient.JAVA
  - Client$ClientUpdateListener.class
  - Client.class
  - createGraph.JAVA
  - createGraph.class
  - LVCTest.JAVA

- LVCTest.class

- **Directory of D:\Source Code\3. LVC Unit Testing Versionr_C++**
  - IPRO_313_LVC_UNIT_TEST.zip

- **Directory of D:\Source Code\4. Data Translator_C++**
  - Translator.zip

- **Directory of D:\Supporting Documents**

- **Directory of D:\Supporting Documents\Design Group**
  - ArcaX White Paper.pdf
  - EXEGY(FPGA).pdf
  - FAST Specification.pdf
  - FAST Tech Reveiw.pdf
  - FAST USERS Guide.pdf
  - FIX Functionality Matrix.pdf
  - HIGH-PERFORMANCE COMPUTING(HPC)(2).ppt
  - High-Performance Market Data Distribution.pdf
  - POC - Phase1b Results
  - POC - Phase1A Results
  - PS3.ppt
  - STAC Report.pdf
  - The 8 Requirements of Realtime Stream Processing.doc
  - Trading at Light Speed.ppt

- **Directory of D:\Supporting Documents\Hardware Group**
  - [IPRO_313]_Hardware_Group_Competing_Products_Oct_16.doc
  - [IPRO_313]_Hardware_Group_Competing_Products_Oct_16.ppt
  - DySPAN_paper.pdf
  - Exegy Ticker Plant.pdf
  - exegy-infiniband-RHEL4client-rev1.01.pdf
  - FInal Report_hardware_v2.doc
  - Hardware results.doc
  - hardware-testing-outline.ods
  - intel-quickassist-white-paper.pdf
  - IPRO313_Hardware_Nov_27.ppt
  - RMDS6-RHEL4-Caneland-XeonX7350-Ethernet-ver1.0.pdf
  - RMDS and Realtick info. from web_2.doc
  - RMDS-IBMx3650-DualCoreXeon-Eth-SLES9-rev1[1].0.pdf
  - Wombat-OPRA-Cisco-DAL-InfiniBand-rev1.0.pdf

- **Directory of D:\Supporting Documents\Software Group**
  - A Flexible Architecture for Statistical Learning and Data Mining.pdf
  - Architecture for Accessing Data Streams on the Grid.pdf
  - aurora_chapter.pdf
  - FPGA.pdf
  - IPRO313_2nd_Presentation_Status_Report_200701008_v0.11.ppt
  - IPRO313_3rd_Presentation_Status_Report_200701024_v0.2.ppt
  - IPRO313_Software Analysis Report_20070829_v0.1.doc
  - IPRO313_Software Analysis Report_20071024_v0.1.doc
  - IPRO313_Software_Presentation_20070829_v0.2.ppt
  - Kdb+Tick.doc

- lookup3.c
- NYSEArca_Equities_Clearing.pdf
- RealTickUserManual.pdf
- revisions.pdf
- schedule_b_standard.pdf
- sparsehash-0.8-1.zip
- STAC Report on Exegy.ppt
- STAC_report_on_Exegy.pdf
- streaming-for-dummies.pdf
- superfasthash.c
- Technology White Paper (April 2006).pdf
- test statistics.xls
- test statsistic2.xls
- trading_floor_architecture
- uthash-1.2.tar
- XTP2.0web.pdf
- yunkiThesis.pdf
-