

# ETL Queues for Active Data Warehousing

Alexis Karakasidis

Panos Vassiliadis

Evaggelia Pitoura



Dept. of Computer Science  
University of Ioannina

# Forecast

- **Active DW** refers to the idea of updating the DW with new contents asap
- We demonstrate that we can employ **queue theory to predict the behavior** of the refreshment process of an Active DW
- We discuss **implementation issues** in order to achieve **minimal system overhead** and **high freshness of data**

# Contents

- Problem description
- System Architecture & Theoretical Analysis
- Experiments
- Conclusions and Future Work

# Contents

- Problem description
- System Architecture & Theoretical Analysis
- Experiments
- Conclusions and Future Work

# Active Data Warehousing

- Traditionally, data warehouse refreshment has been performed **off-line**, through Extracttion-Transformation-Loading (**ETL**) software.
- **Active Data Warehousing** refers to a new trend where **data warehouses are updated as frequently as possible**, to accommodate the high demands of users for fresh data.
- **Issues** that come up:
  - How to **design** an Active DW?
  - How can we **implement** an Active DW?

# Issues and Goals of this paper

- **Smooth upgrade** of the software at the (**legacy**) source
    - Minimal modification of the software configuration at the source side.
  - **Minimal overhead** of the source system
- 
- **No data losses** are allowed
  - **Maximum freshness** of data
    - The response time for the transport, cleaning, transformation and loading of a new source record to the DW should be small and predictable
- 
- **Stable interface** at the warehouse side
    - The architecture should scale up with respect to the number of sources and data consumers at the DW
    - If possible, cover issues like checkpointing, index maintenance, etc.

# Contributions

- We **set up the architectural framework** and the issues that arise for the case of active data warehousing.
- We **develop the theoretical framework** for the problem, by employing queue theory for the prediction of the performance of the system.
  - We provide **a taxonomy for ETL tasks** that allows treating them as black-box tasks.
  - Then, **standard queue theory techniques can be applied** for the design of an ETL workflow.
- We provide **technical solutions** for the implementation of our reference architecture, achieving the aforementioned goals
- We prove our results through extensive **experimentation**.

# Related work

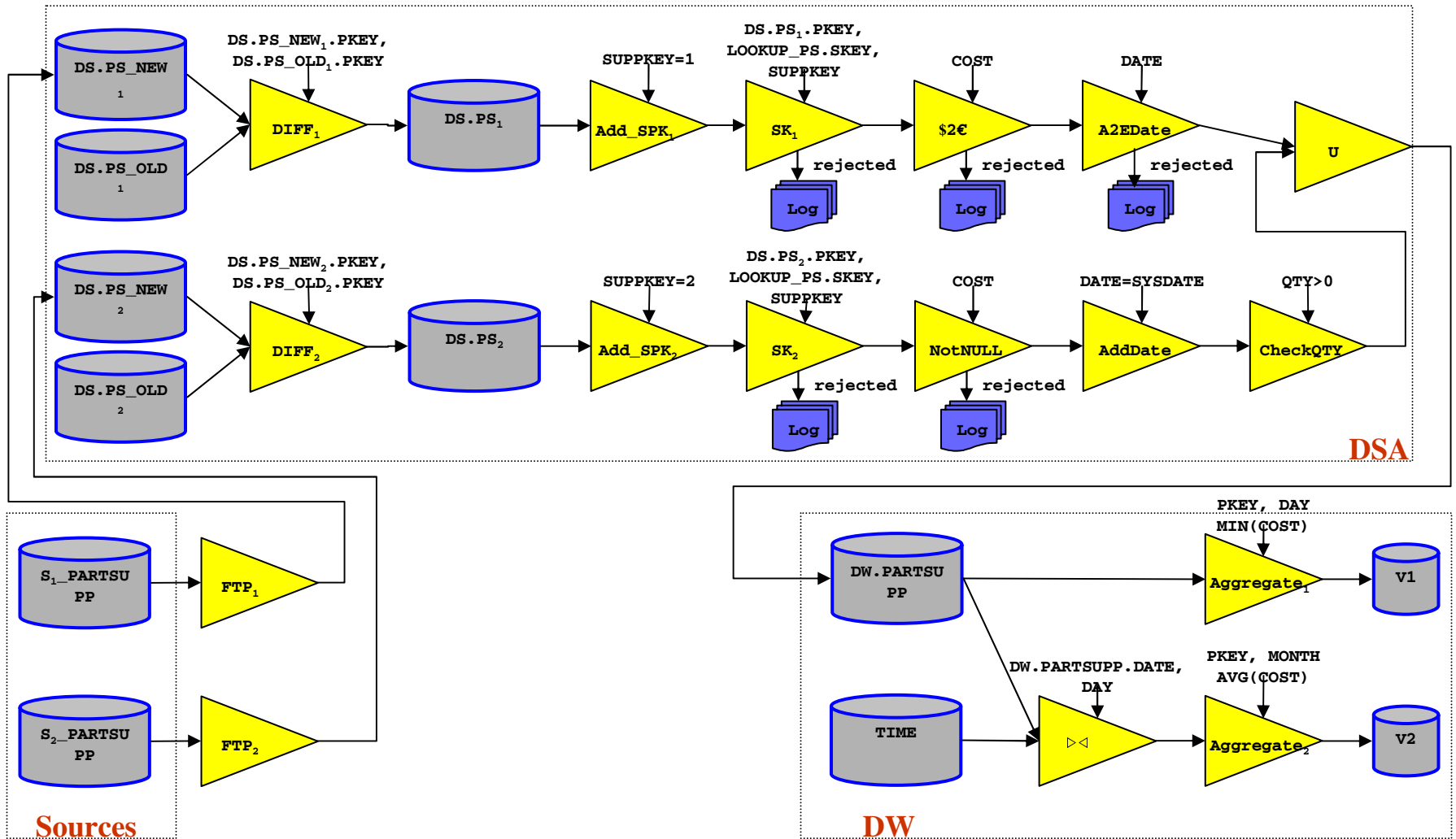
- Obviously, work in the field of ETL is related
  - must be customized for active DW
- Streams, due to the nature of the data
  - still, all R.W. is on continuous queries, no updates
- Huge amount of work in materialized view refreshment
  - orthogonal to our problem
- Web services
  - due to the fact that in our architecture, the DW exports W.S.'s to the sources



# Contents

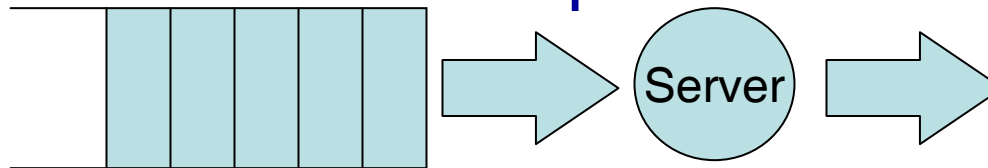
- Problem description
- System Architecture & Theoretical Analysis
- Experiments
- Conclusions and Future Work

# ETL workflows



# Queue Theory for ETL

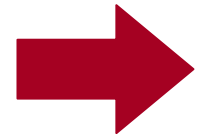
- We can model various kinds of ETL transformations as queues, which we call **ETL queues**
- Each queue has an incoming arrival rate  $\lambda$  and a mean service time  $1/\mu$



- **Little's Law:**  $N = \lambda * T$
- Based on their statistical properties, queue characteristics can be estimated
  - E.g., **M/M/1 queue** (Poisson arrivals)
    - Mean response time  $W = 1/(\mu - \lambda)$
    - Mean queue length  $L = \rho / (1 - \rho)$ ,  $\rho = \lambda / \mu$

# Queue Theory for ETL

- Queues can be combined to form **queue networks**
- **Jackson networks**: networks where each queue can be solved independently (under reasonable constraints)
- How can we use queue theory to predict the behavior of the Active Data Warehouse?

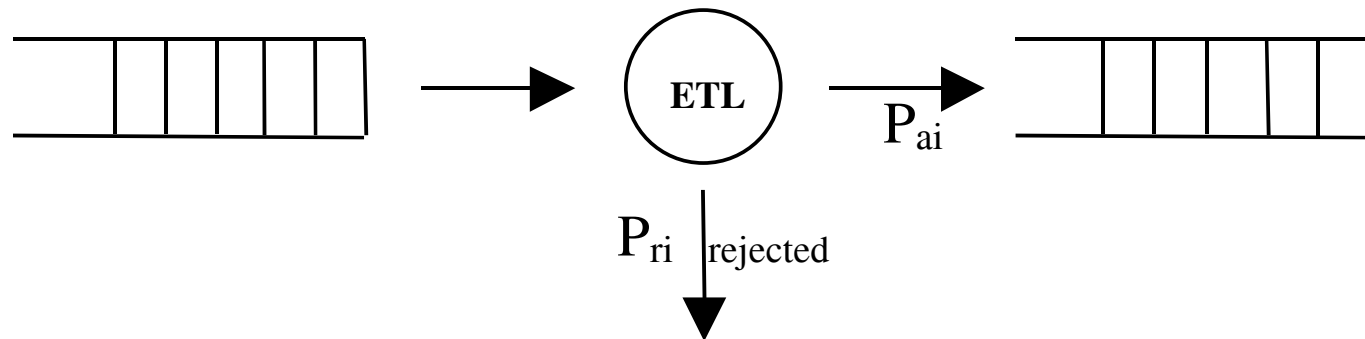


# How to predict the behavior of the Active Data Warehouse

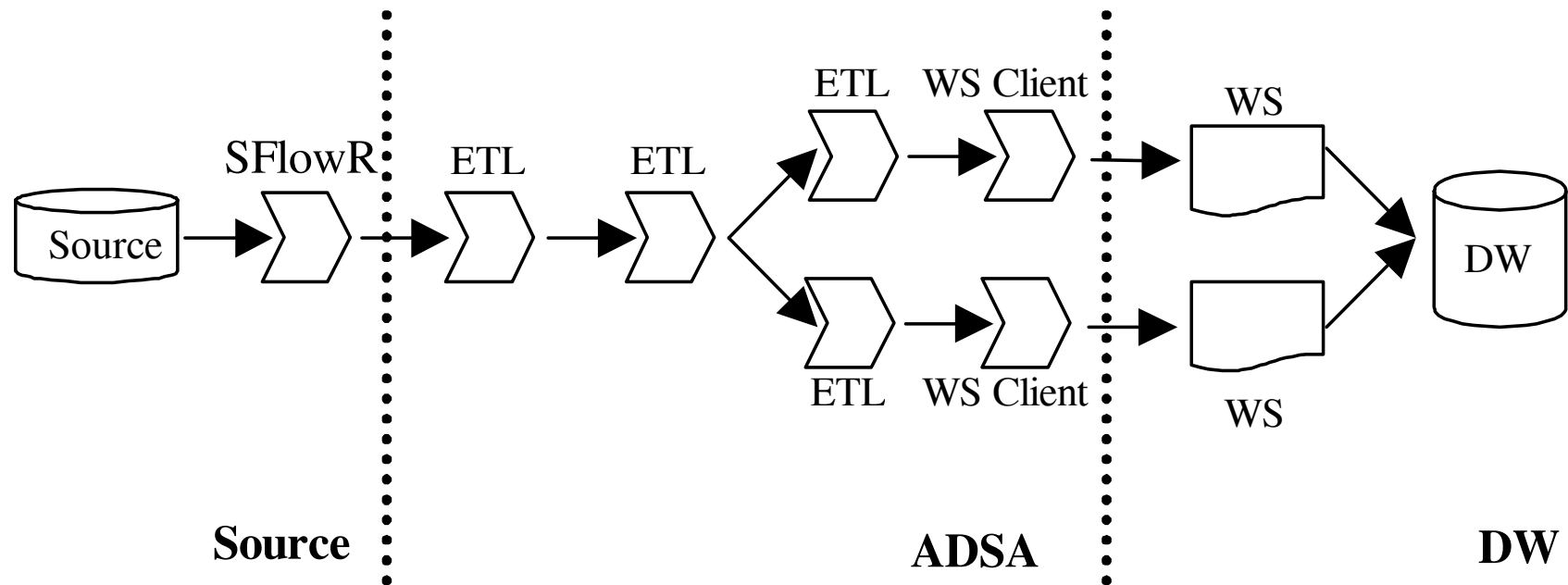
1. Compose ETL queues in a Jackson network to simulate the implementation of the **Active Data Staging Area (ADSA)**
2. Then, solve the Jackson network and relate the parameters of ADSA, specifically:
  - **Source arrival rate** (i.e., rate or record production at the source)
  - **Overall service time** (i.e., time that a record spends in the ADSA)
  - **Mean queue length** (i.e., no. of records in the network)

# Taxonomy of ETL transformations

- Filters
- Transformers
- Binary Operators
  
- Generic model



# System Architecture



# System Architecture

- **Source Flow Regulator (SFlowR)**: compiles changes in blocks and propagates them
- **Active Data Staging Area (ADSA)**
  - Performs all the ETL tasks
  - Can act as a regulator for the flow towards the DW
- **ETL queues**: periodically check their queue and process incoming blocks
- **WS-client**: each WS needs a client to invoke it (also acting as a WS-FlowR)
- **Web Service**: one per DW table.
- **WS/WS-client**: can be used for load-balancing, checkpointing, ...



# Contents

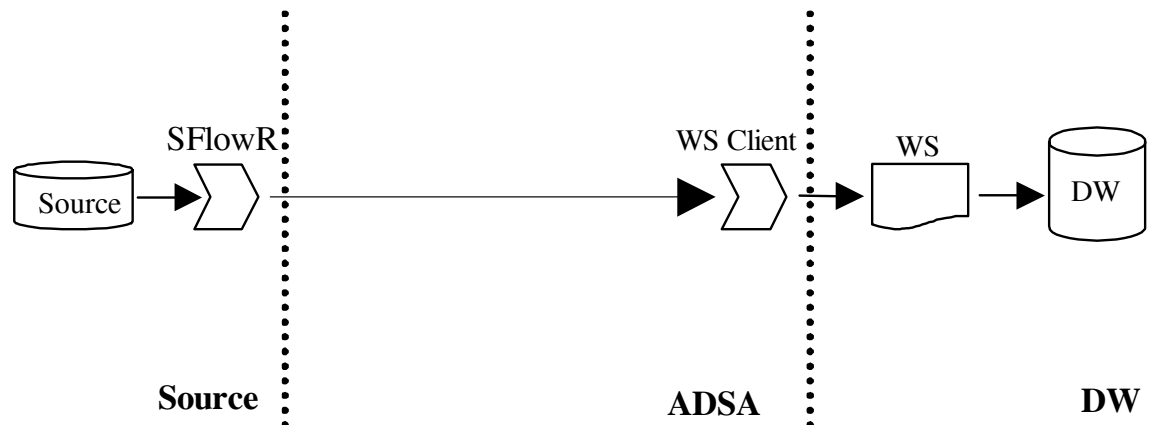
- Problem description
- System Architecture & Theoretical Analysis
- Experiments
- Conclusions and Future Work

# Experimentation environment

- Source: an application in C that uses an ISAM library
- ADSA implemented in Sun JDK 1.4
- Web Services platform:
  - Apache Axis 1.1 [AXIS04]
  - Xerces XML parser
  - Apache Tomcat 1.3.29
- DW implemented over MySQL 4.1
- Configuration:
  - Source: PIII 700MHz with 256MB memory, SuSE Linux 8.1
  - DW: Pentium 4 2.8GHz with 1GB memory, Mandrake Linux, ADSA included
  - Department's LAN for the network
- Source operates at full capacity

# First set of experiments

- A first set of experiments over a simple configuration, to determine fundamental architectural choices
- Issues
  - Smooth upgrade of the source software
  - UDP vs TCP
  - Source Overhead
  - Data delay
  - Topology



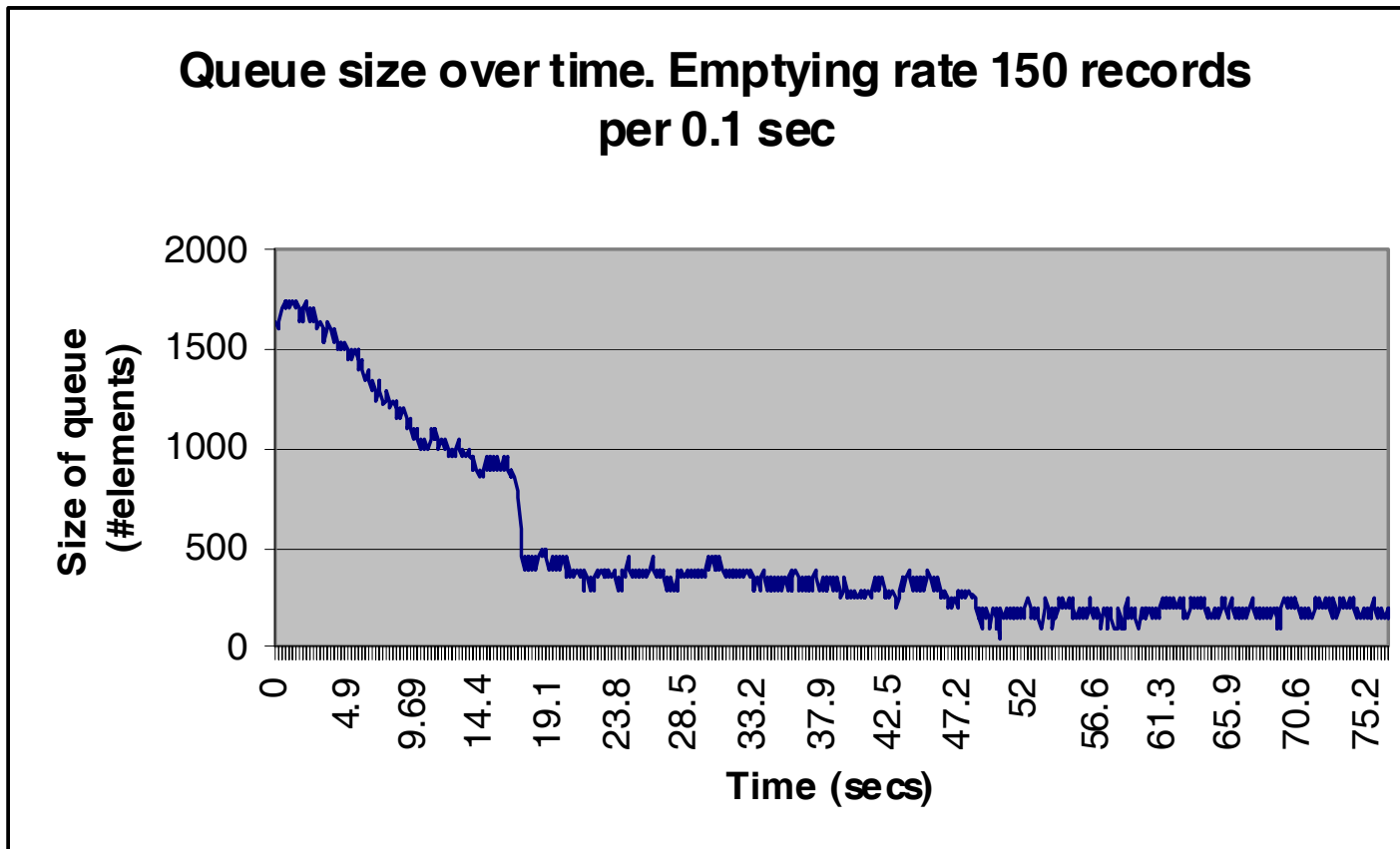
# Experimentation results

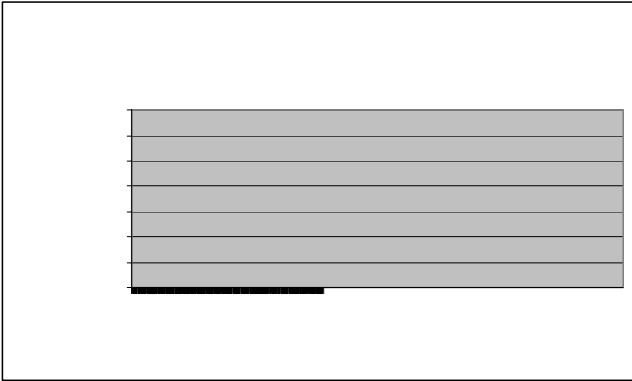
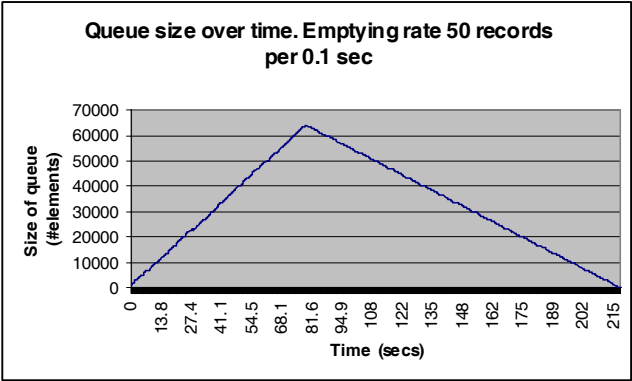
- **Smooth upgrade**: not more than 100 lines of **library** code modified
- UDP resulted in 35% data loss, due to ADSA overflow => **TCP a clear choice**
- **Source overhead** is highly dependent on **row blocking**:
  - Source overhead is 1.7% with a source flow regulator, vs **34% without**
  - WS mode (blocking vs non-blocking) has no effect
  - Medium size packets seem to work better

# Data Freshness

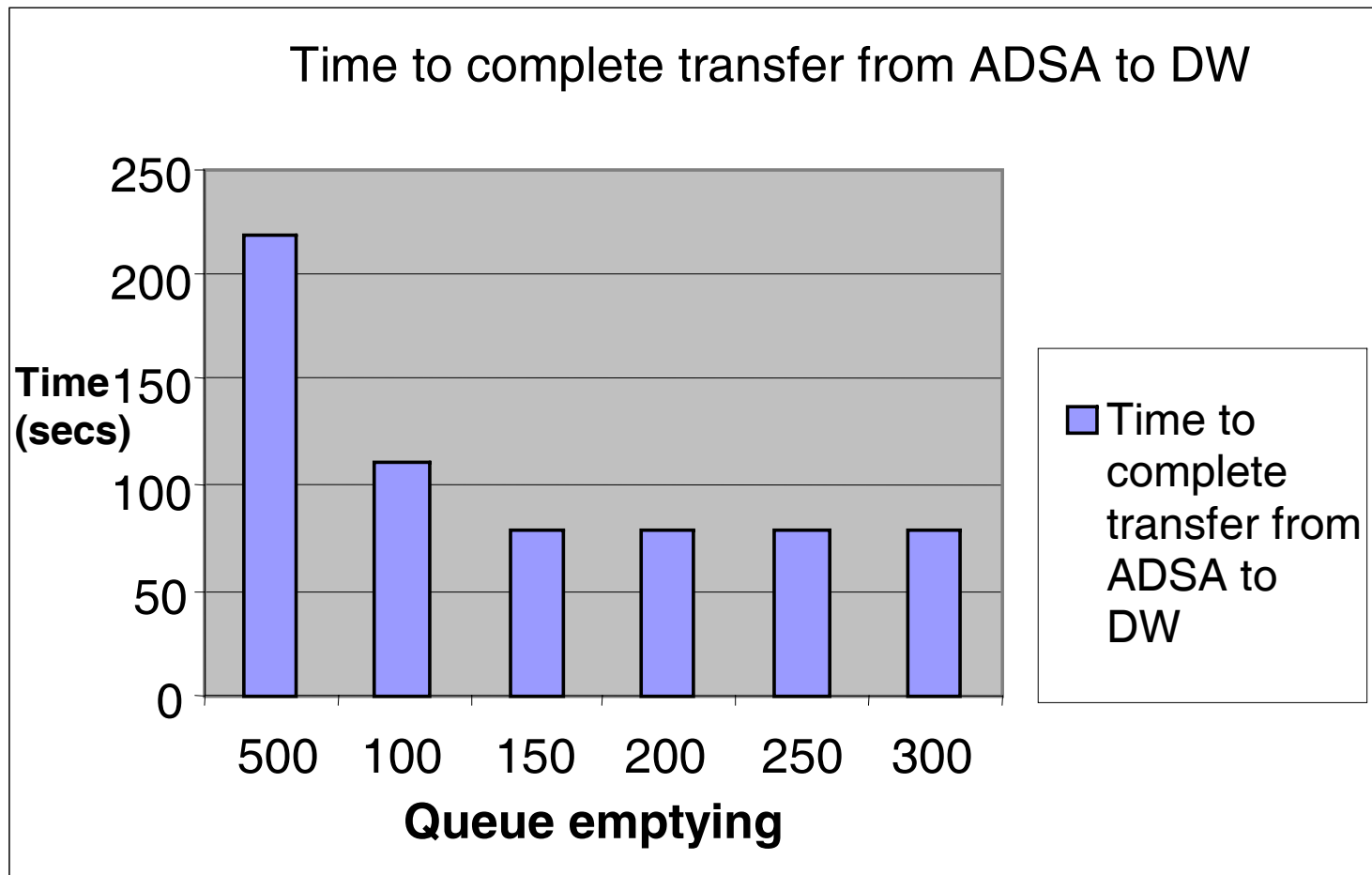
- We count the **time to carry all records from source to DW**
- We empty the ADSA with 3 policies:
  - Immediate transport
  - We simulate a slower ADSA by removing 50, 100, 150, 200, 250 and 300 records from the queue every 0.1 sec
  - We remove 500, 1000, 1500, 2000, 2500 and 3000 records every 1 sec
  - Source max rate is about 1250 records / sec
- **Findings:**
  - **Small package sizes result in small delays**
  - There is a threshold (the source rate) underneath which the queue explodes
  - We can achieve data freshness time equal to data insertion time when we **continuously empty a small size queue**

# Data Freshness





# Data Freshness

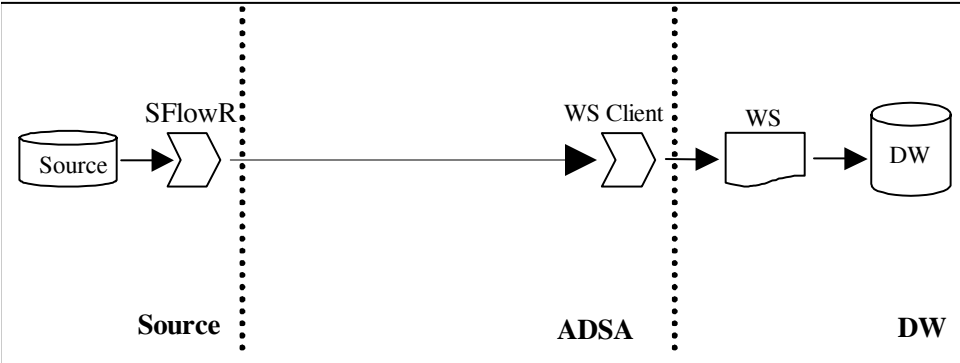




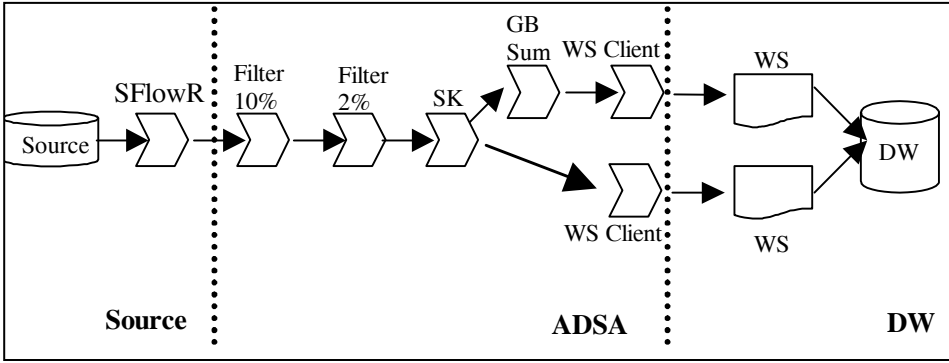
# Experiments including transformation scenarios

- We enrich the previous configuration with several **ETL activities in the ADSA**
- Based on the previous, we have fixed:
  - **2-tier architecture**, ADSA at the DW
  - Source Flow Regulation with **medium size packages**
  - **TCP** for network connection
  - Non-blocking calling of DW WS's

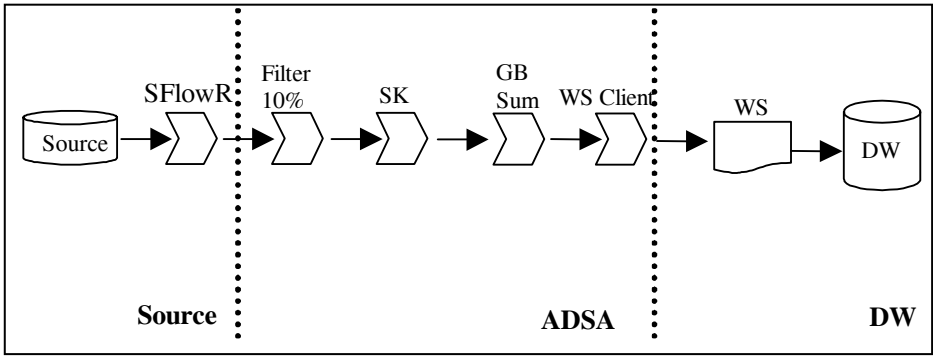
# Scenarios to measure data freshness



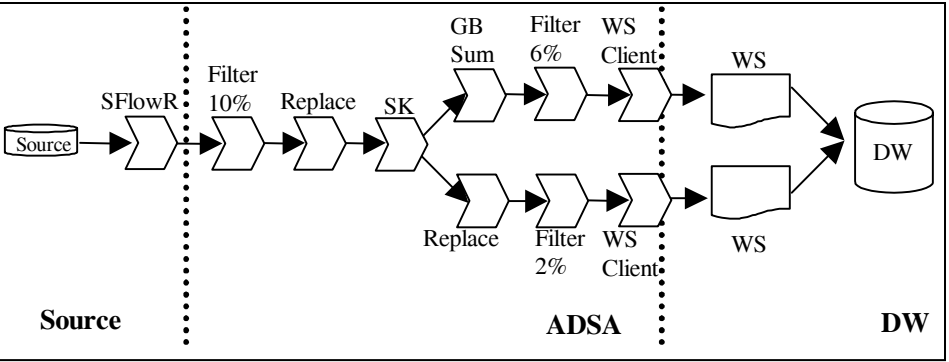
(a)



(c)



(b)



(d)

# Goals of the experiments

- **Steadiness** of the system
  - System is steady whenever service rate is higher than arrival rate; transient effects disappear
- **Source overhead**
  - Medium size blocking is still a winner
- **Throughput** for ADSA
  - The ADSA is only one packet behind the source
  - Avg. delay per row  $\sim 0.9$  msec for all scenarios
- Success of **theoretical prediction**
  - Half a packet underestimation

# Contents

- Problem description
- System Architecture & Theoretical Analysis
- Experiments
- Conclusions and Future Work

# Conclusions

- We can **employ queue theory to predict the behavior of an Active ETL process**
- We have **proposed an architectural configuration with**
  - Minimal source overhead
  - No effect on the source due to the operation of an ADSA
  - No packet losses, due to the usage of TCP
  - Small delay in the ADSA, especially if row blocking in medium size blocks is used

# Future Work

- Combine our configuration with results in the **optimization of ETL processes** (ICDE'05)
- **Fault tolerance**
- Experiment with **higher client loads** at the warehouse side
- Scale-up the **number of sources** involved

# Thank you!



# Backup Slides



IQIS'05 17 June 2005, Baltimore MD, USA

# Jackson's Theorem and ETL queues

*Jackson's Theorem.* If in an open network the condition  $\lambda_i < \mu_i \cdot m_i$  holds for every  $i \in \{1, \dots, N\}$  (with  $m_i$  standing for the number of servers at node  $i$ ) then the steady state probability of the network can be expressed as the product of the state probabilities of the individual nodes:

$$\pi(k_1, \dots, k_N) = \pi_1(k_1)\pi_2(k_2)\dots \pi_N(k_N)$$

Therefore, **we can solve this class of networks** in four steps:

- Solve the traffic equations to find  $\lambda_i$  for each queuing node  $i$
- Determine separately for each queuing system  $i$  its steady-state probabilities  $\pi_i(k_i)$
- Determine the global steady-state probabilities  $\pi(k_1, \dots, k_N)$ . Derive the desired global performance measures.
- From step 1, we can derive the mean delay and queue length for each node.

# Source Code Alterations

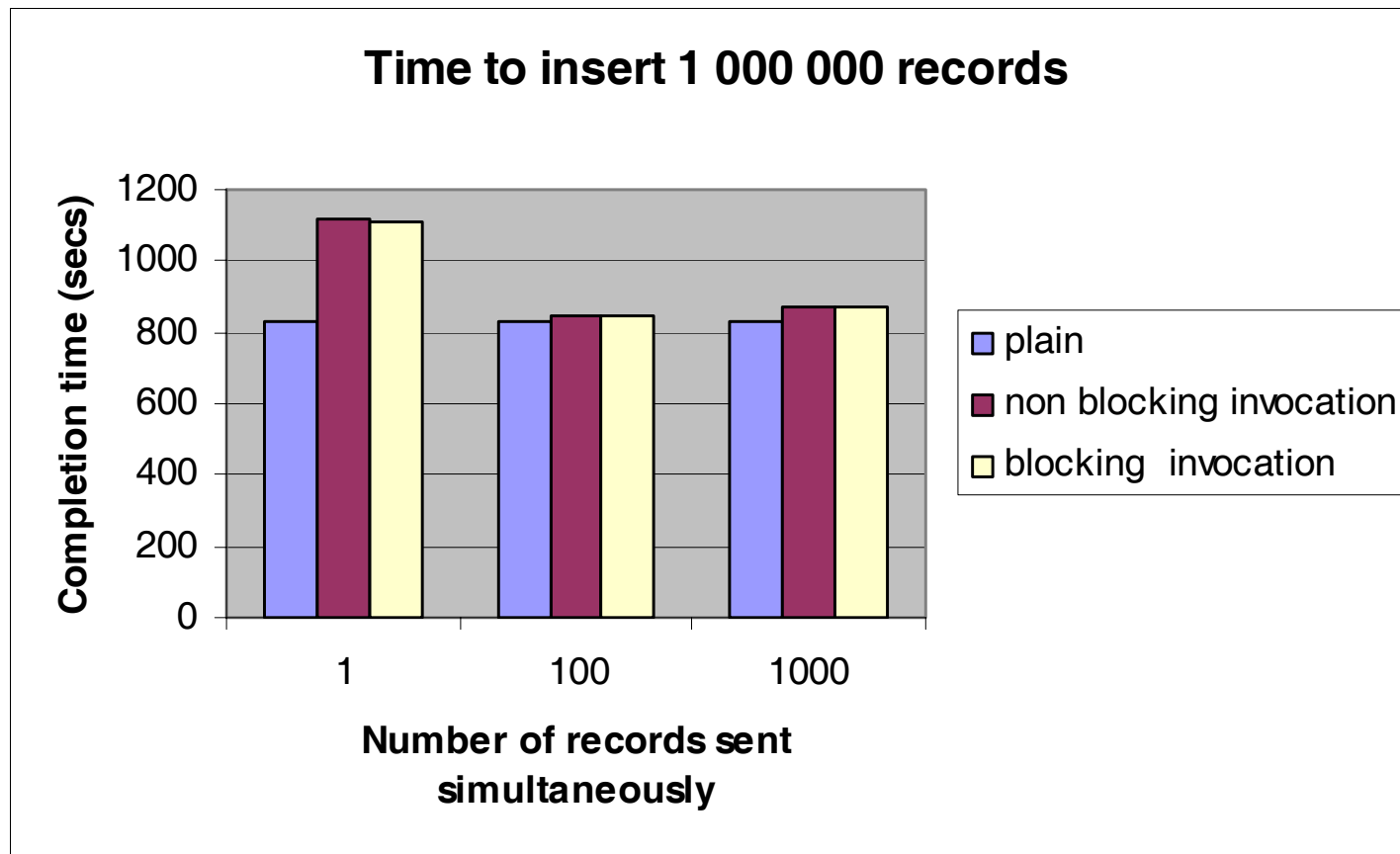
Original Routine	Altered Routine
<pre>Open_isam_File(){     ...     opening_isam_file_commands     ... }</pre>	<pre>Open_isam_File(){     ...     opening_isam_file_commands     ...     if(open==success)         <b>DWFlowR_socket_open()</b> }</pre>
<pre>Write_record_to_File(){     ...     insert_record_commands     ... }</pre>	<pre>Write_record_to_File(){     ...     insert_record_commands     ...     if(write==success)         <b>write_to_SFlowR()</b> }</pre>
<pre>Close_isam_File(){     ...     closing_isam_file_commands     ... }</pre>	<pre>Close_isam_File(){     ...     closing_isam_file_commands     ...     if(close==success)         <b>DWFlowR_socket_close()</b> }</pre>

# First set of experiments

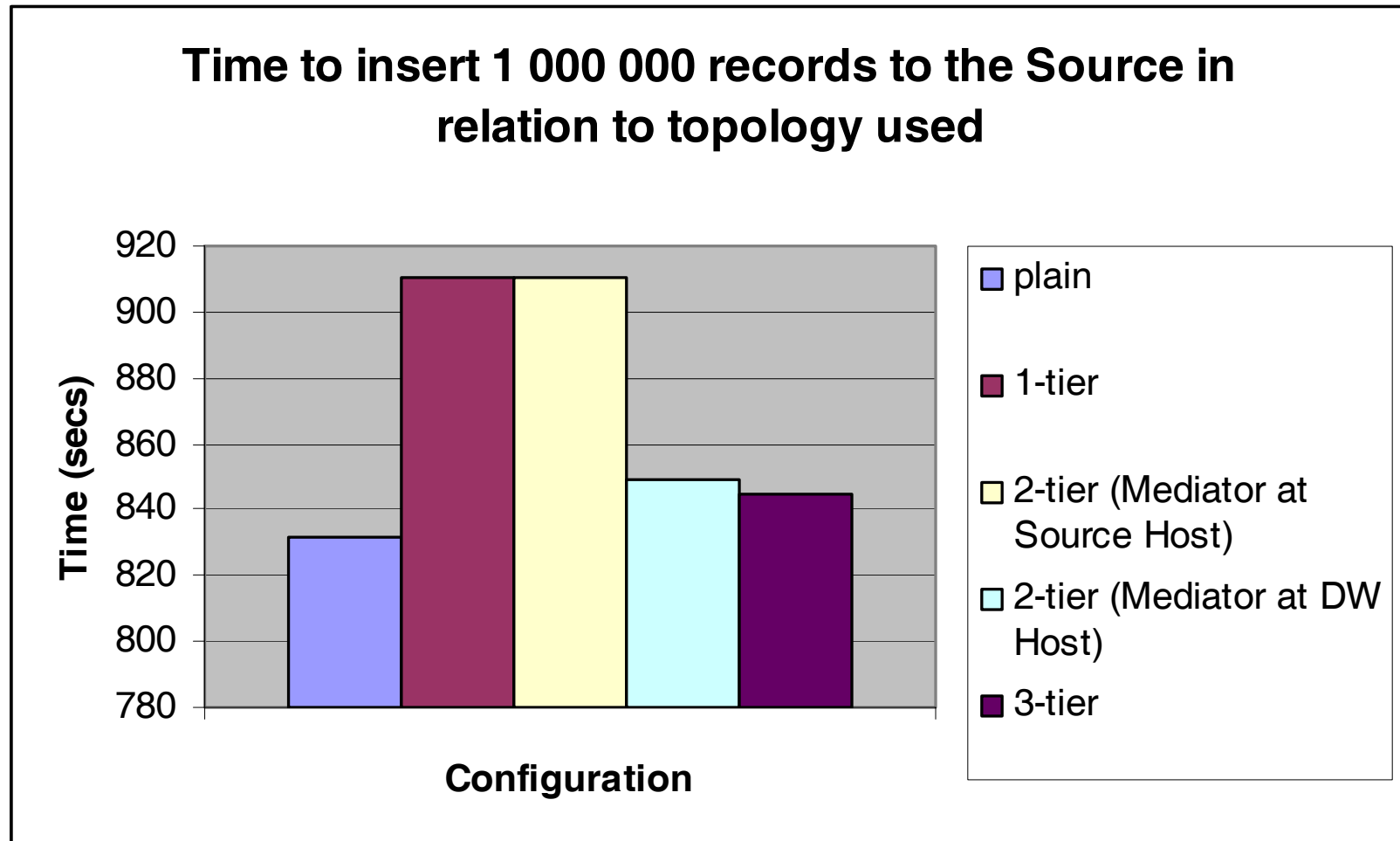
# Data Freshness

- We count the time to carry all records from source to DW
- We empty the ADSA with 3 policies:
  - Immediate transport
  - We simulate a slower ADSA by removing 50, 100, 150, 200, 250 and 300 records from the queue every 0.1 sec
  - We remove 500, 1000, 1500, 2000, 2500 and 3000 records every 1 sec
- Source max rate is about 1250 records / sec
- Findings:
  - Small package sizes result in small delays
  - There is a threshold (the source rate) underneath which the queue explodes
  - We can achieve data freshness time equal to data insertion time when we *continuously* empty a *small size* queue

# Source overhead



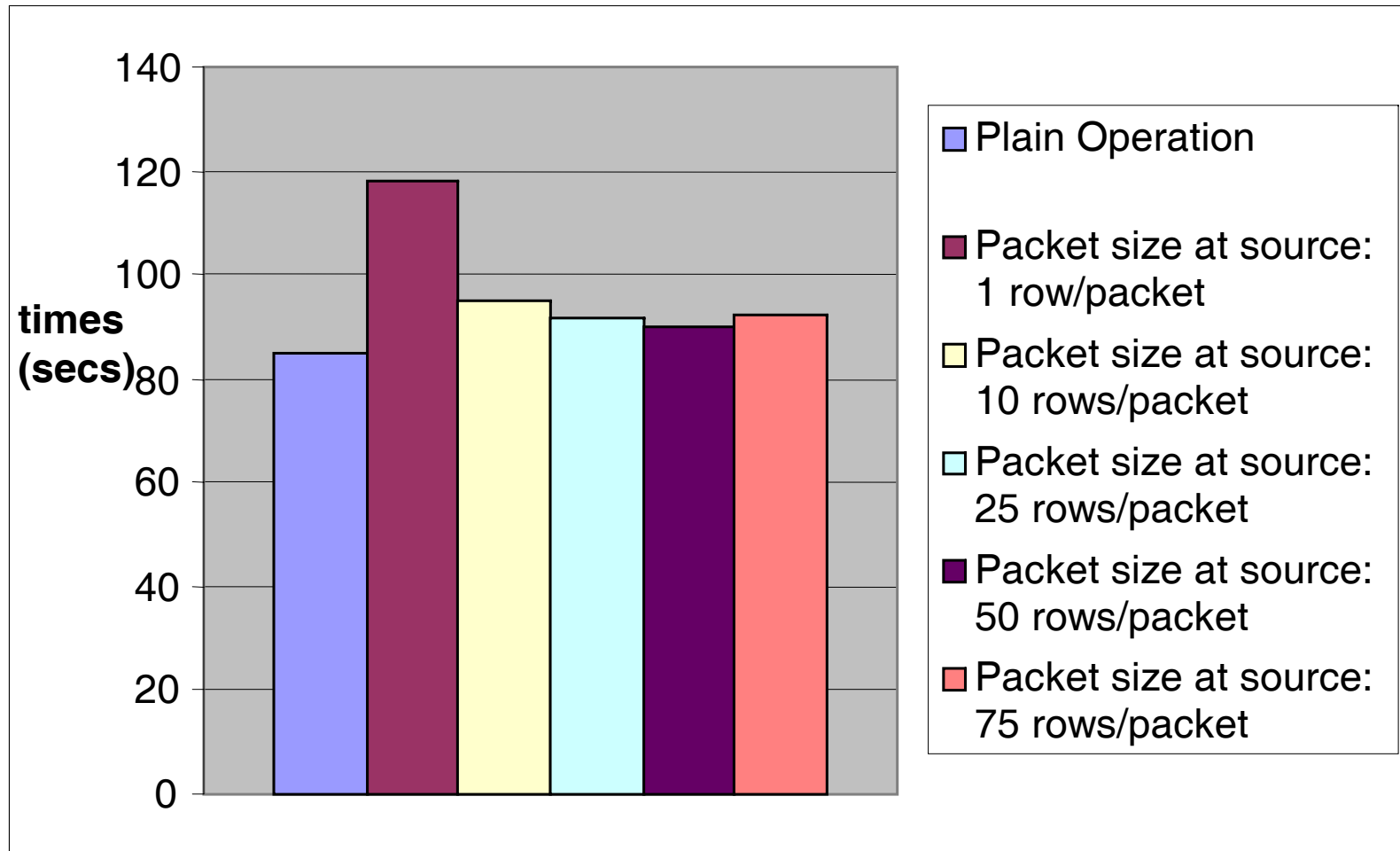
# Topology and source overhead



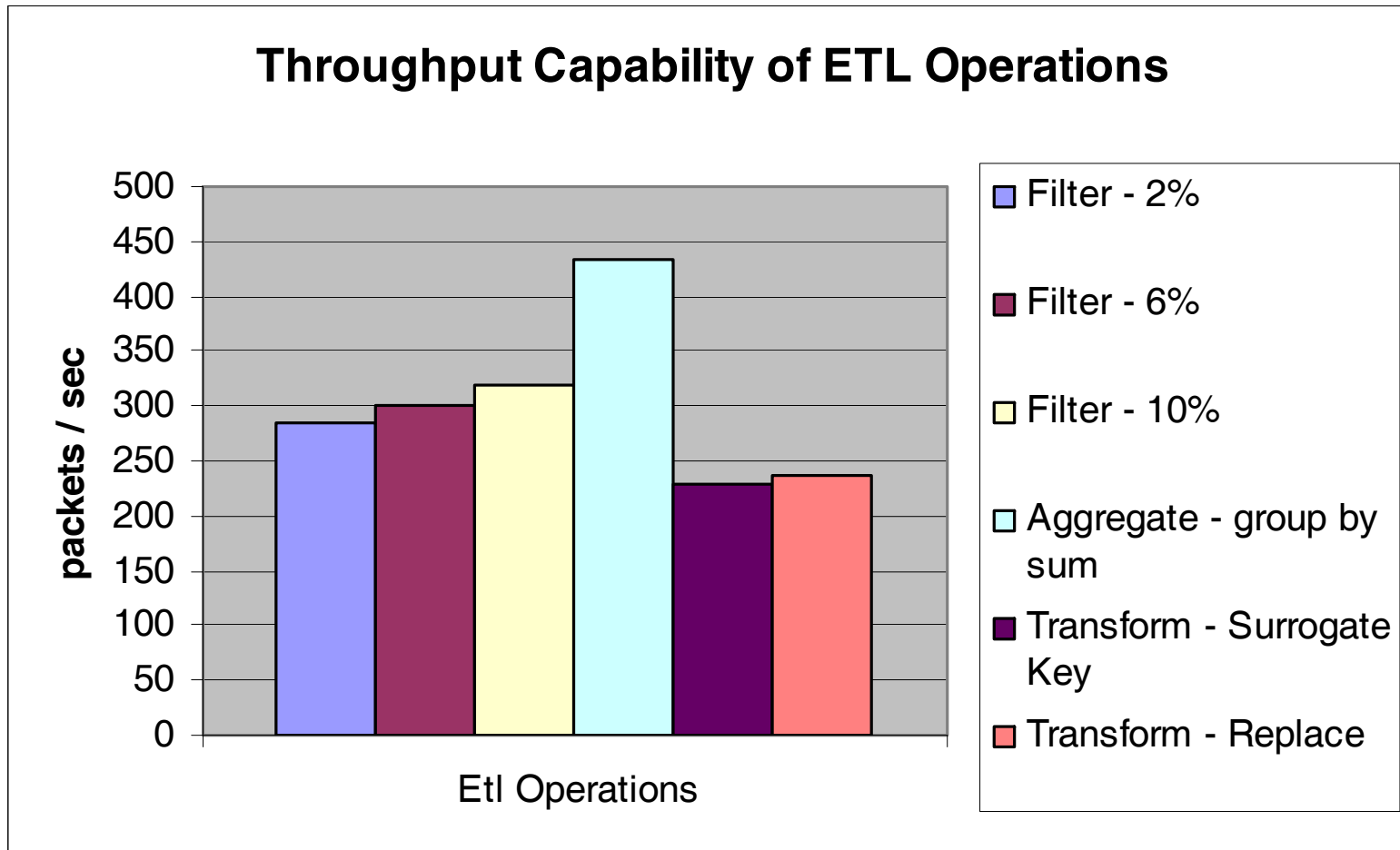
# Second set of experiments



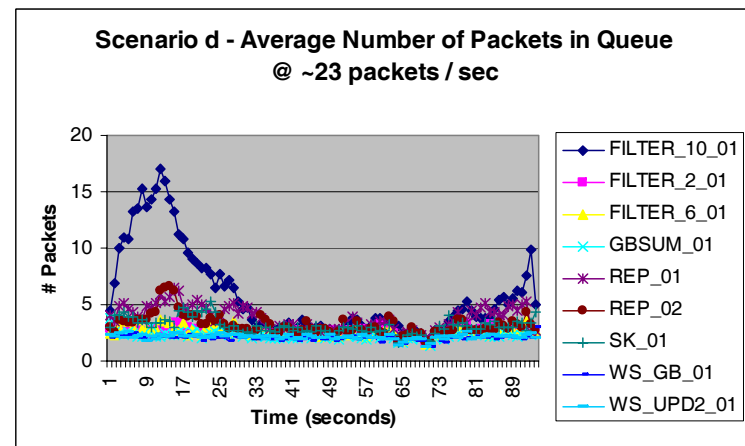
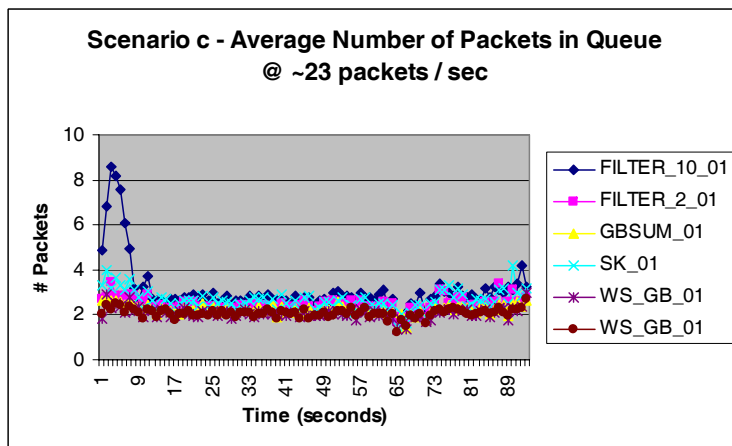
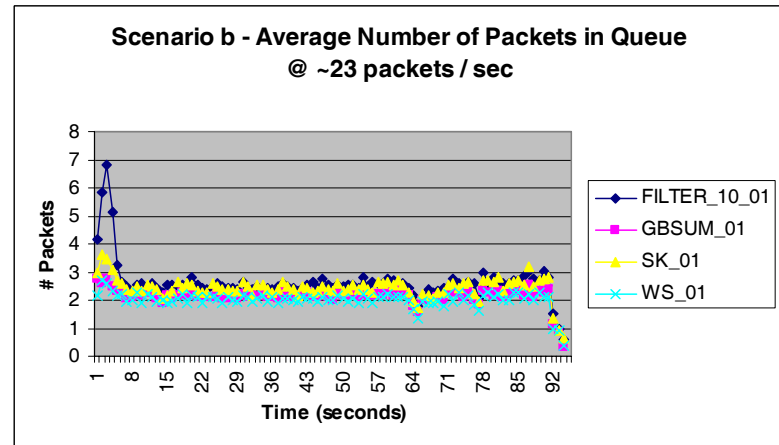
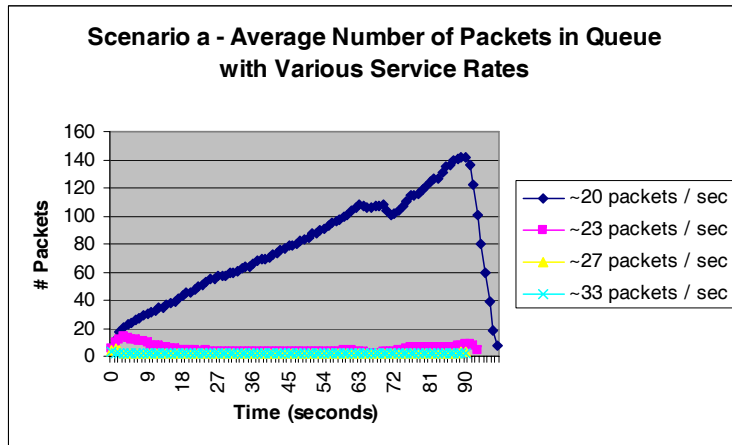
# Source overhead



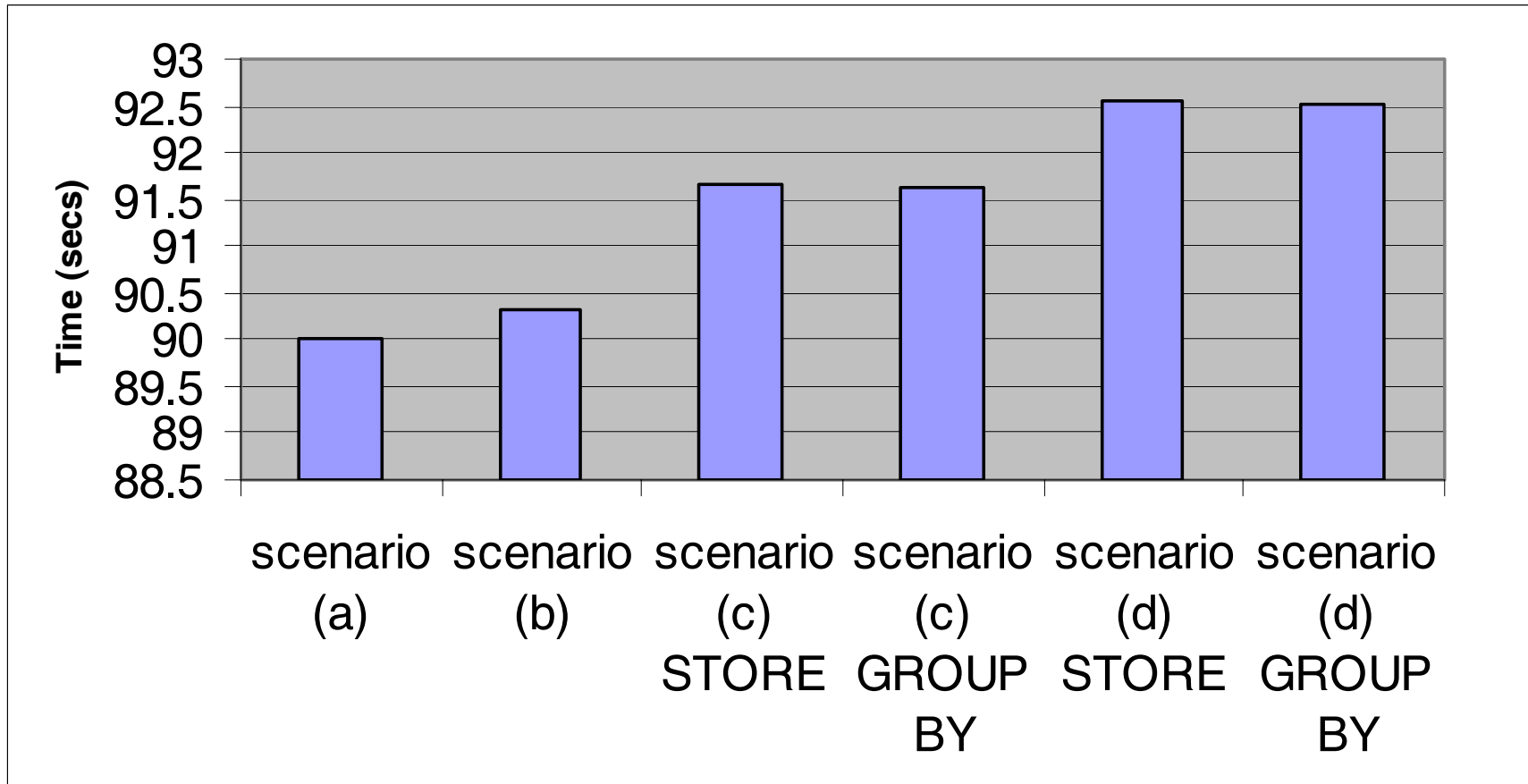
# Throughput for ETL operations



# Scenarios to measure data freshness



# Data Delay



## Theoretical prediction vs. actual measurements of average queue length for scenario (c) in packets

	<b>Measured</b>	<b>Theoretical Prediction</b>	<b>Difference</b>
<b>FILTER_10_01</b>	<b>0.160</b>	<b>0.056</b>	<b>0.104</b>
<b>FILTER_02_01</b>	<b>0.134</b>	<b>0.047</b>	<b>0.087</b>
<b>SK_01</b>	<b>0.154</b>	<b>0.054</b>	<b>0.100</b>
<b>GB_SUM_01</b>	<b>0.137</b>	<b>0.048</b>	<b>0.089</b>
<b>WS_GB</b>	<b>0.091</b>	<b>0.031</b>	<b>0.059</b>
<b>WS_GB_UPD</b>	<b>0.100</b>	<b>0.035</b>	<b>0.066</b>

# Theoretical Predictions and Actual Measurements

- In most cases, we underestimate the actual queue size by half a packet (i.e., 25 records)
- We overestimate the actual queue size when we simulate slow servers, esp. in the combination of large timeouts and large packets
- Reasons for the discrepancies:
  - Simulation of slower rates through timeouts
  - Due to the row-blocking approach, the granule of transport is a single packet