The NetLogger Toolkit

Brian L. Tierney (bltierney@lbl.gov)
Future Technologies Group

(http://www-didc.lbl.gov/NetLogger)
Lawrence Berkeley National Laboratory[1]
Berkeley, CA 94720

1. This work is jointly supported by DARPA - ITO, and by the U. S. Dept. of Energy, Energy Research Division, Mathematical, Information, and Computational Sciences office, under contract DE-AC03-76SF00098 with the University of California.

Overview

- **The Problem:**

  When building high-speed network-based distributed services, we often observe unexpectedly low network throughput and/or high latency - the reasons for which are usually not obvious.

  The bottlenecks can be in any of the following components:

  - **the applications**
  - **the operating systems**
  - **the device drivers, the network adapters on either the sending or receiving host (or both)**
  - **the network switches and routers, and so on**

- **The Solution:**

  Highly instrumented systems with precision timing information and analysis tools (NetLogger)

Outline

- **Introduction: Performance Analysis and Monitoring using the NetLogger Toolkit**
- **NetLogger Components:**
  - **Event Log Generation Library**
  - **Event Log Visualization Tools**
- **Background: The Magic Gigabit Network Testbed, the DPSS, and TerraVision**
- **NetLogger Analysis of the DPSS**
- **Work in Progress**
  - **Event Log Management Agents**

Motivation

There are virtually no behavioral aspects of widely distributed applications that can be taken for granted - they are fundamentally different from LAN-based distributed applications.

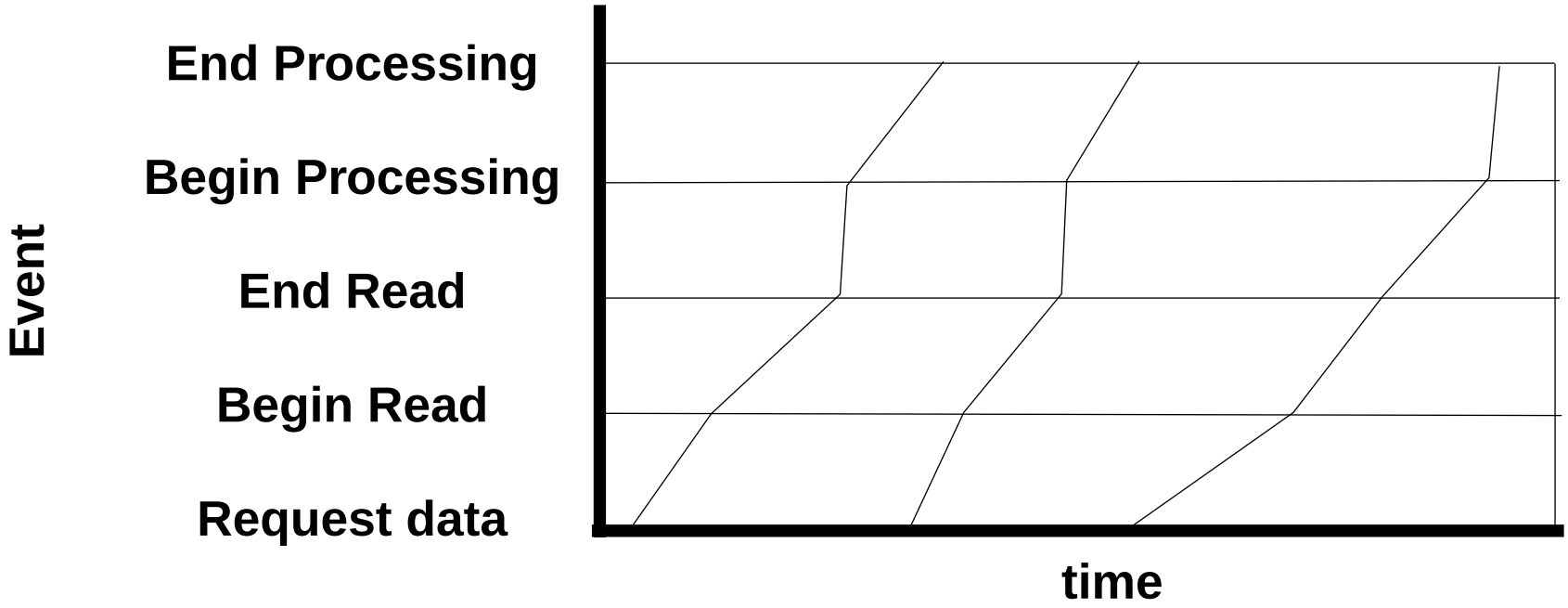
- **Techniques that work in the lab frequently do not work in a wide-area network environment (even a testbed network)**

To characterize the wide area environment we have developed a methodology for detailed, *end-to-end, top-to-bottom monitoring* and analysis of every significant event involved in distributed systems data interchange.

- **This has proven invaluable for isolating and correcting performance bottlenecks, and even for debugging distributed parallel code.**

- **Network performance tools such at *ttcp* and *netperf* are somewhat useful, but don't model real distributed applications, which are complex, bursty, and have more than one connection in and/or out of a given host at one time.**
- **To address this, we have developed the NetLogger Toolkit: a set of tools to aid in creating graphs that trace a data request throughout a distributed system.**
- **This allows us to determine exactly what is happening within this complex system.**
- **Using "life-lines" to visualize the data flow is the key to easy interpretation of the results. (see figure)**

NetLogger Event “Life-lines”



Netlogger Components

- Use a common log format
- Application libraries for generating NetLogger Messages
  - Can send log messages to:
    - file
    - syslogd
    - host/port (netlogd)
  - C, C++ and Java are currently supported
- Event Visualization tools
- Management Agents
- Modified Unix network and OS monitoring tools to log "interesting" events using the same log format
    - *netstat*, *vmstat*, and tcpdump modified output results in the NetLogger log format

NetLogger log format:

We are using the IETF draft standard Universal Logger Message (ULM) format:

- a list of “field=value” pairs
- required fields: DATE, HOST, PROG, and LVL
  - LVL is the severity level (Emergency, Alert, Error, Usage, etc.)
- followed by optional user defined fields

NetLogger adds these required fields:

- NL.EVNT, a unique identifier for the event being logged. i.e.:
  DPSS_SERV_IN, VMSTAT_USER_TIME, NETSTAT_RETRANSSEG
- NL.SEC, and NL.USEC, which are the seconds and microseconds from the Unix *gettimeofday* system call

Sample NetLogger ULM event:

```
DATE=19980430133038 HOST=foo.lbl.gov
PROG=testprog LVL=Usage NL.EVNT=SEND_DATA
NL.SEC=893968238 NL.USEC=55784 SEND.SZ=49332
```

This says program named *testprog* on host *foo.lbl.gov* performed event named SEND_DATA, size = 49332 bytes, at the time given.

User-defined data elements (any number) are used to store information about the logged event - for example:

NL.EVNT=SEND_DATA SEND.SZ=49332

- **the number of bytes of data sent**

NL.EVNT=NETSTAT_RETRANSSEGS NS.RTS=2

- **the number of TCP retransmits since the previous event**

NetLogger API:

Open calls:

```
NLhandle   *lp = NULL;

/* log to a local file */
lp = NetLoggerOpen(method, program_name, log_filename,
                   NULL, 0);
/* log to syslog */
lp = NetLoggerOpen(method, program_name, NULL, NULL, 0);

/* log to “netlogd” on the specified host/port */
lp = NetLoggerOpen(method, program_name, NULL, hostname,
                   DPSS_NETLOGGER_PORT);
```

Write the log event:

```
NetLoggerWrite(lp, "EVENT_NAME",  "F1=%d F2=%d F3=%d
        F4=%.2f", data1, data2, string, fdata);
```

Sample Code:

```
/* log to a local file */
lp = NetLoggerOpen(method, progname, log_filename, NULL, 0);

while (!done)
{
    NetLoggerWrite(lp, "EVENT_START", "TEST.SIZE=%d", size);

    /* perform the task to be monitored */
    done = do_something(data, size);

    NetLoggerWrite(lp, "EVENT_END");
}
NetLoggerClose(lp);
```

Network Time Protocol

- For NetLogger timestamps to be meaningful, all systems clocks must be synchronized.
- NTP is used to synchronize time of all hosts in the system.
  - NTP is from Dave Mills, U. of Delaware (http://www.eecis.udel.edu/~ntp/)
  - All hosts run *xntpd*, which synchronizes the clocks of each host both to GPS-based time servers and to each other
  - This allows us to synchronize the clocks of all hosts to within about 250 microseconds of each other, but...
    - systems have to stay up for a significant length of time for the clocks to converge to 250 μs
    - best to have a time server on the same network as all hosts
    - many different sys admins (harder to synchronize than clocks)
  - In practice, clock synchronization of 1ms is good enough

NTP

- **Purpose of NTP**
  - **conveys timekeeping information from the primary servers to other time servers via the Internet**
  - **cross-checks clocks and mitigates errors due to equipment or propagation failures**
- **Must have NTP running on one or more primary servers, and on a number of local-net hosts, acting as secondary time servers**
- **Host time servers will synchronize via another peer time server, based on the following timing values:**
  - **those determined by the peer relative to the primary reference source of standard time**
  - **those measured by the host relative to the peer**
- **NTP provides not only precision measurements of offset and delay, but also definitive maximum error bounds, so that the user interface can determine not only the time, but the quality of the time as well.**

NetLogger Visualization Tools

Exploratory, interactive analysis of the log data has proven to be the most important means of identifying problems.

We have developed a tool called *nlv* (NetLogger Visualization).

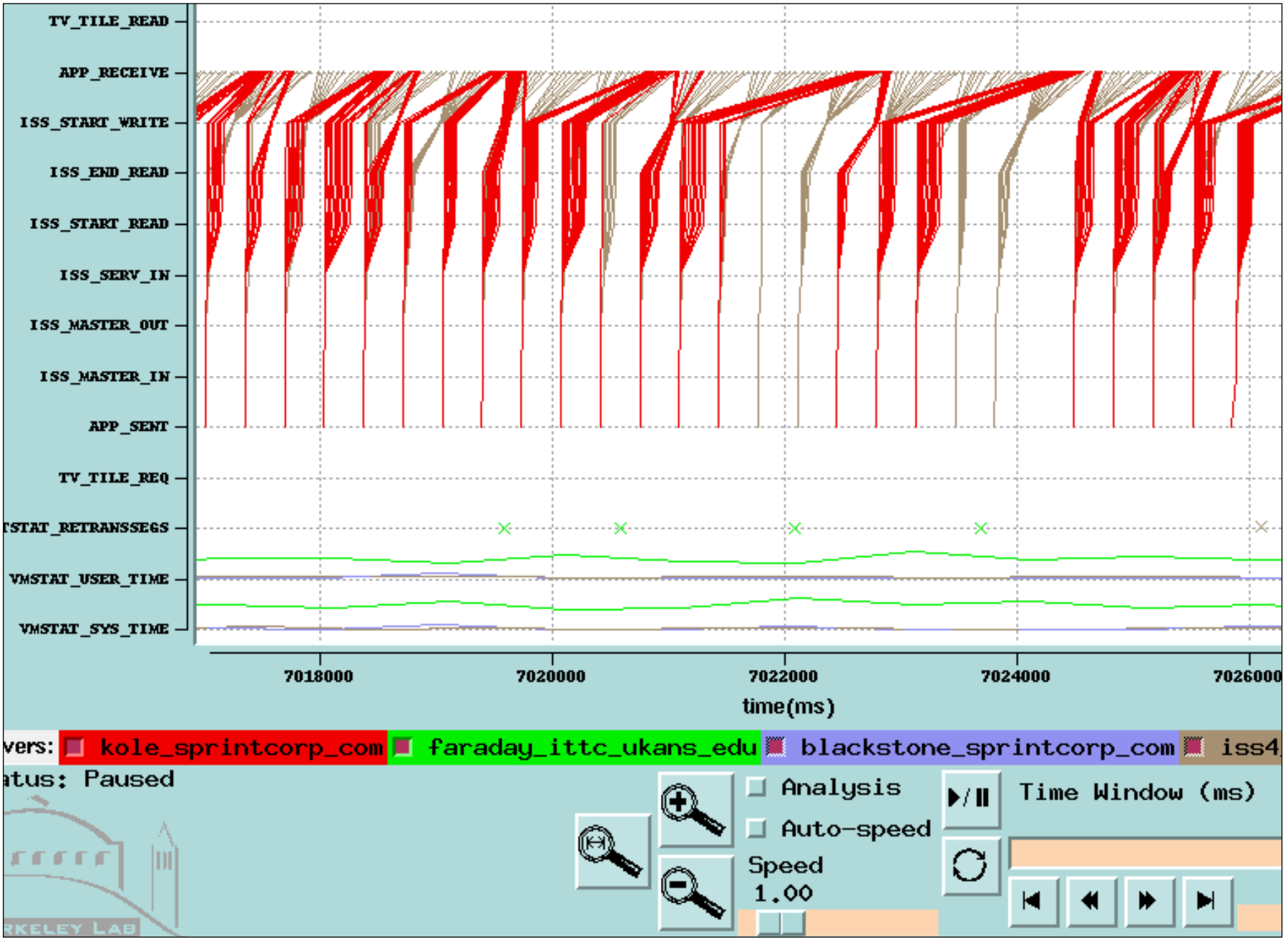
***nlv* functionality:**

- **can display several types of NetLogger events at once**
- **user configurable: which events to plot, and the type of plot to draw (lifeline, load-line, or point)**
- **play, pause, rewind, slow motion, zoom in/out, and so on**
- ***nlv* can be run post-mortem, or in "real-time"**

Other NetLogger tools to analyze log files:

- ***perl* scripts to extract information from log files**
- ***gnuplot* to graph the results**

NLV


TV_TILE_READ
APP_RECEIVE
ISS_START_WRITE
ISS_END_READ
ISS_START_READ
ISS_SERV_IN
ISS_MASTER_OUT
ISS_MASTER_IN
APP_SENT
TV_TILE_REQ
TSTAT_RETRANSSEGS
VMSTAT_USER_TIME
VMSTAT_SYS_TIME
7018000
7020000
7022000
7024000
7026000
time(ms)
vers:
kole_sprintcorp_com
faraday_ittc_ukans_edu
blackstone_sprintcorp_com
iss4
tus: Paused
Analysis
Auto-speed
Speed
1.00
Time Window (ms)

The Distributed-Parallel Storage System (DPSS)

Background:

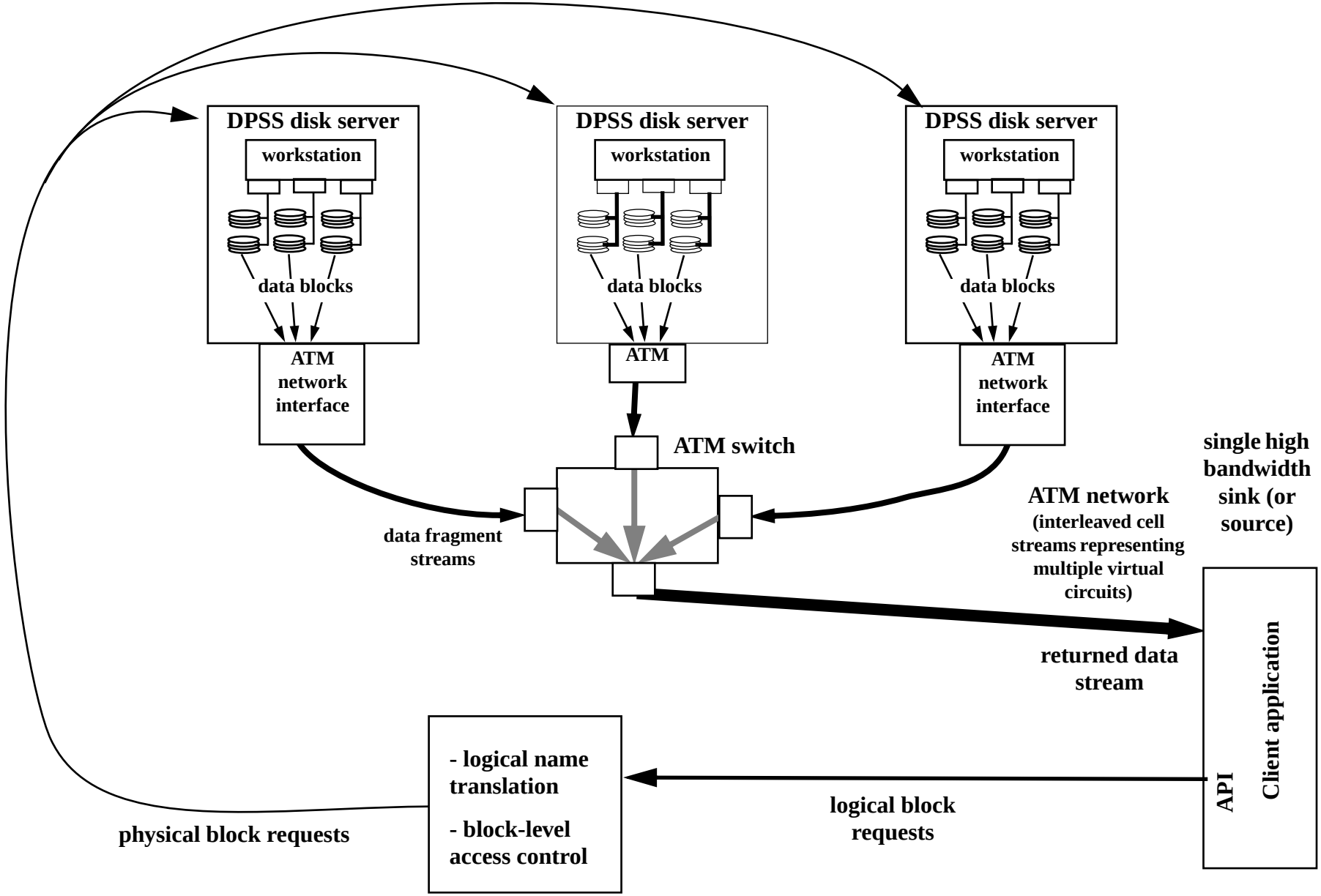
- DPSS was developed as part of the DARPA-sponsored MAGIC Gigabit Network Testbed (see: http://www.magic.net).
- The prototype high-speed application for the DPSS was TerraVision, developed at SRI.
- TerraVision uses tile images and digital elevation models to produce a 3D visualization of landscape.
- DARPA's primary interest in the DPSS and TerraVision was to stress test new ATM OC-3 (and now OC-12) WANs.

DPSS

The DPSS is a collection of wide area distributed disk servers which operate in parallel to provide high-speed, logical block level access to large data sets.

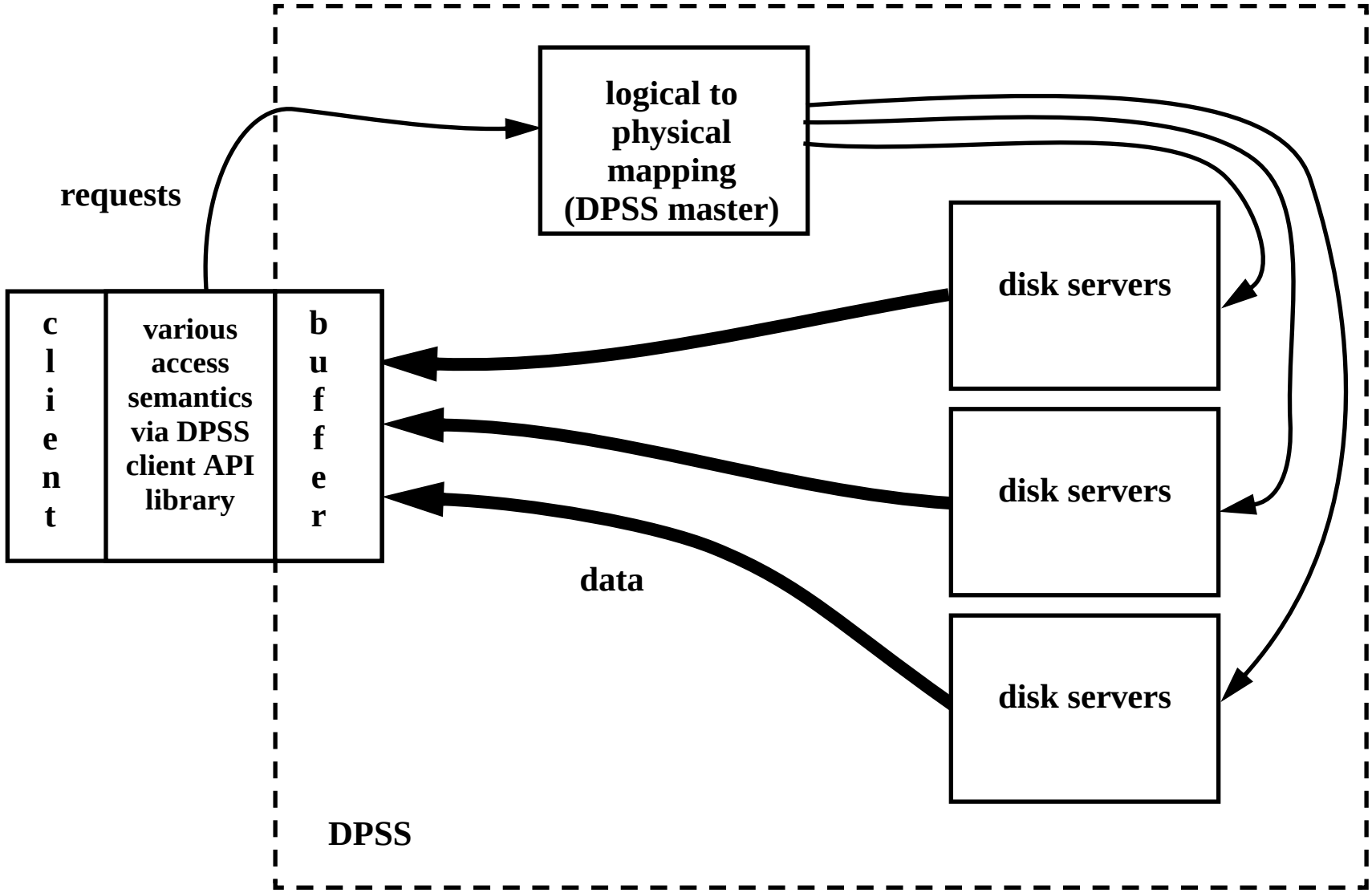
- **Operated primarily as a network-based cache**
- **Built from low-cost commodity hardware components**
- **Implementation has zero memory copies of data blocks, and is all user level code**
- **Runs on Solaris, IRIX, DEC Unix, Linux, FreeBSD, Solaris X86**

DPSS



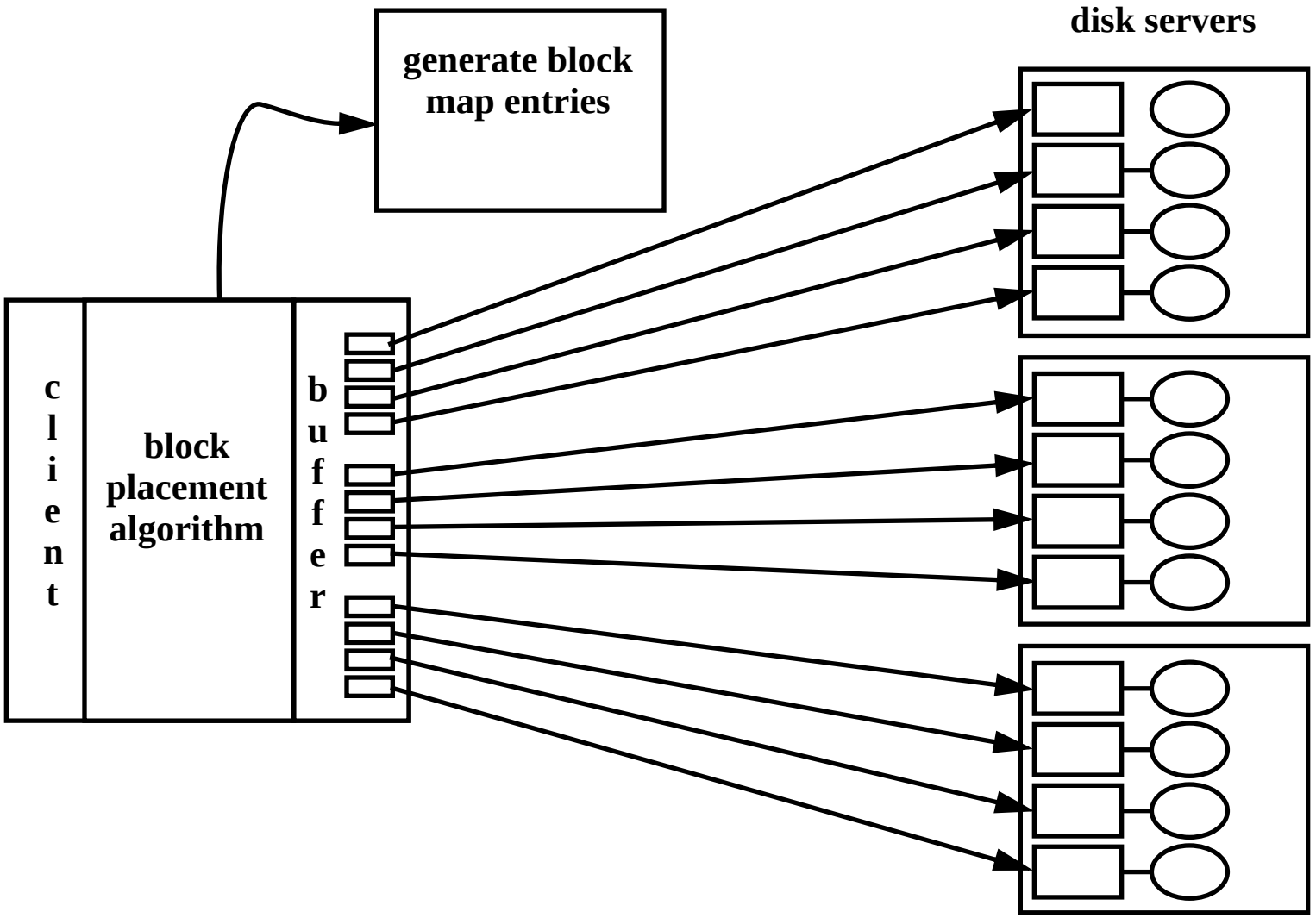
Architecture for Distributed-Parallel Storage System

DPSS



Distributed-Parallel Storage System Model (Reading)

DPSS



DPSS model for high-speed writing

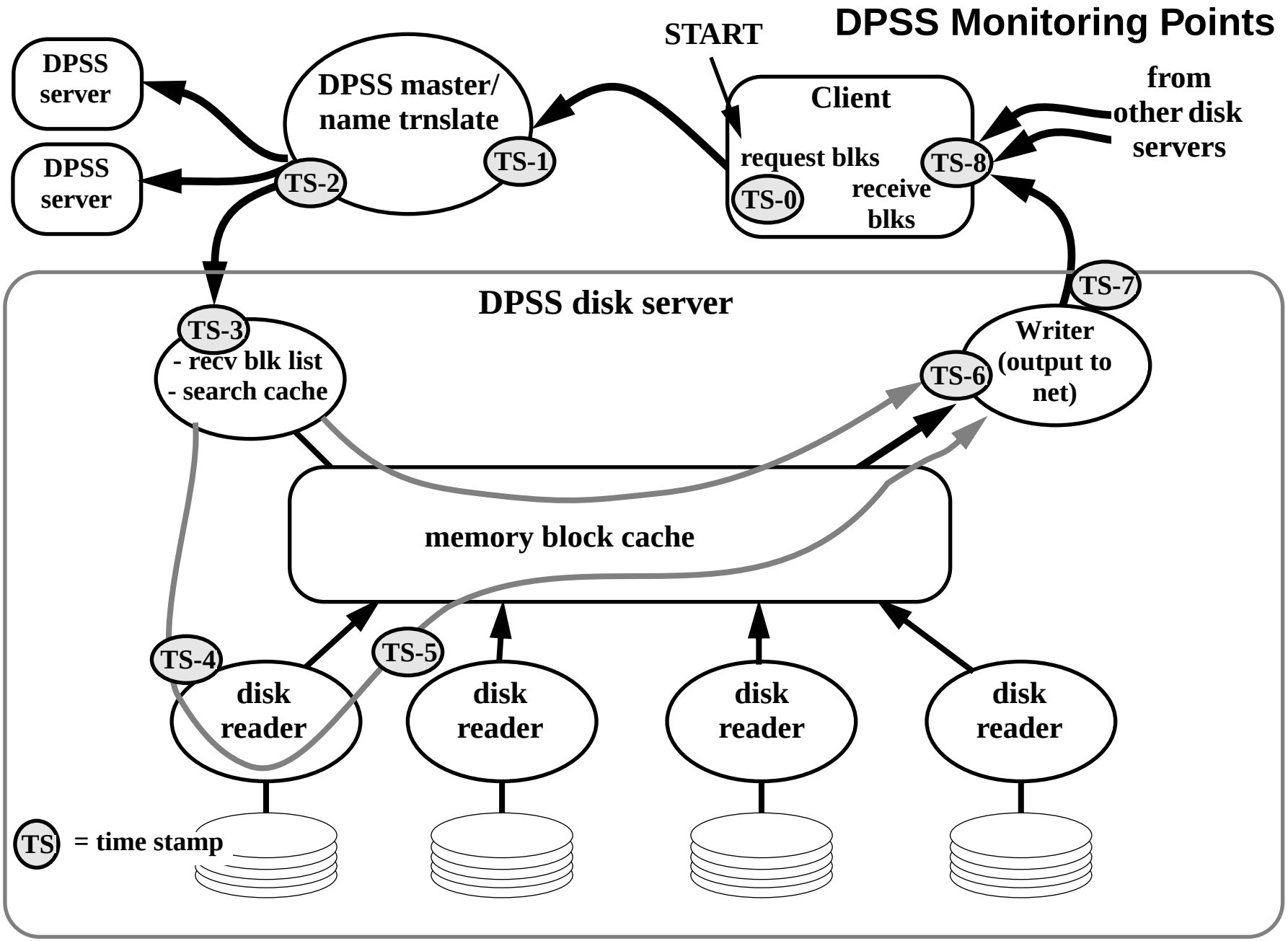
Typical DPSS implementation

- 4 - 5 UNIX workstations (e.g. Sun Ultra I, Pentium 200)
- 4 - 6 fast-SCSI disks on multiple 2 - 3 SCSI host adaptors
- a high-speed network (e.g.: ATM or 100 Mbit ethernet)

♦ This configuration can deliver an aggregated data stream to an application at about 500 Mbits/s (>60 MBy/s) using these relatively low-cost, “off the shelf” components by exploiting the parallelism provided by approximately five hosts, twenty disks, ten SCSI host adaptors, and five network interfaces.

- **DPSS Monitoring Points (see Figure).**

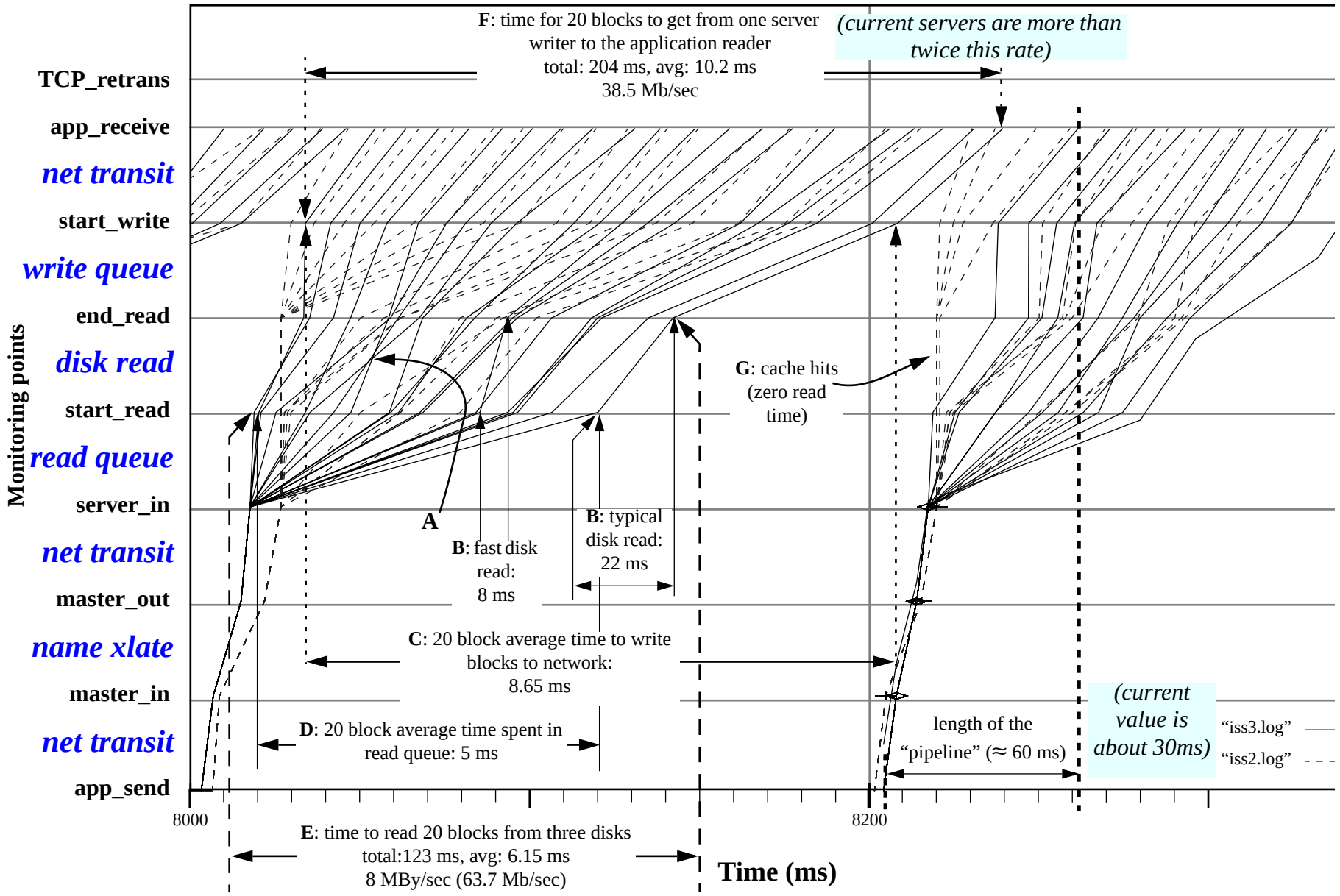
DPSS


DPSS Monitoring Points
START
DPSS server
DPSS server
DPSS master/ name trnslate
TS-1
TS-2
Client
request blks
TS-0
receive blks
TS-8
from other disk servers
DPSS disk server
TS-3
- recv blk list
- search cache
TS-7
Writer (output to net)
TS-6
memory block cache
TS-4
TS-5
disk reader
disk reader
disk reader
disk reader
TS = time stamp

NetLogger Analysis of the DPSS

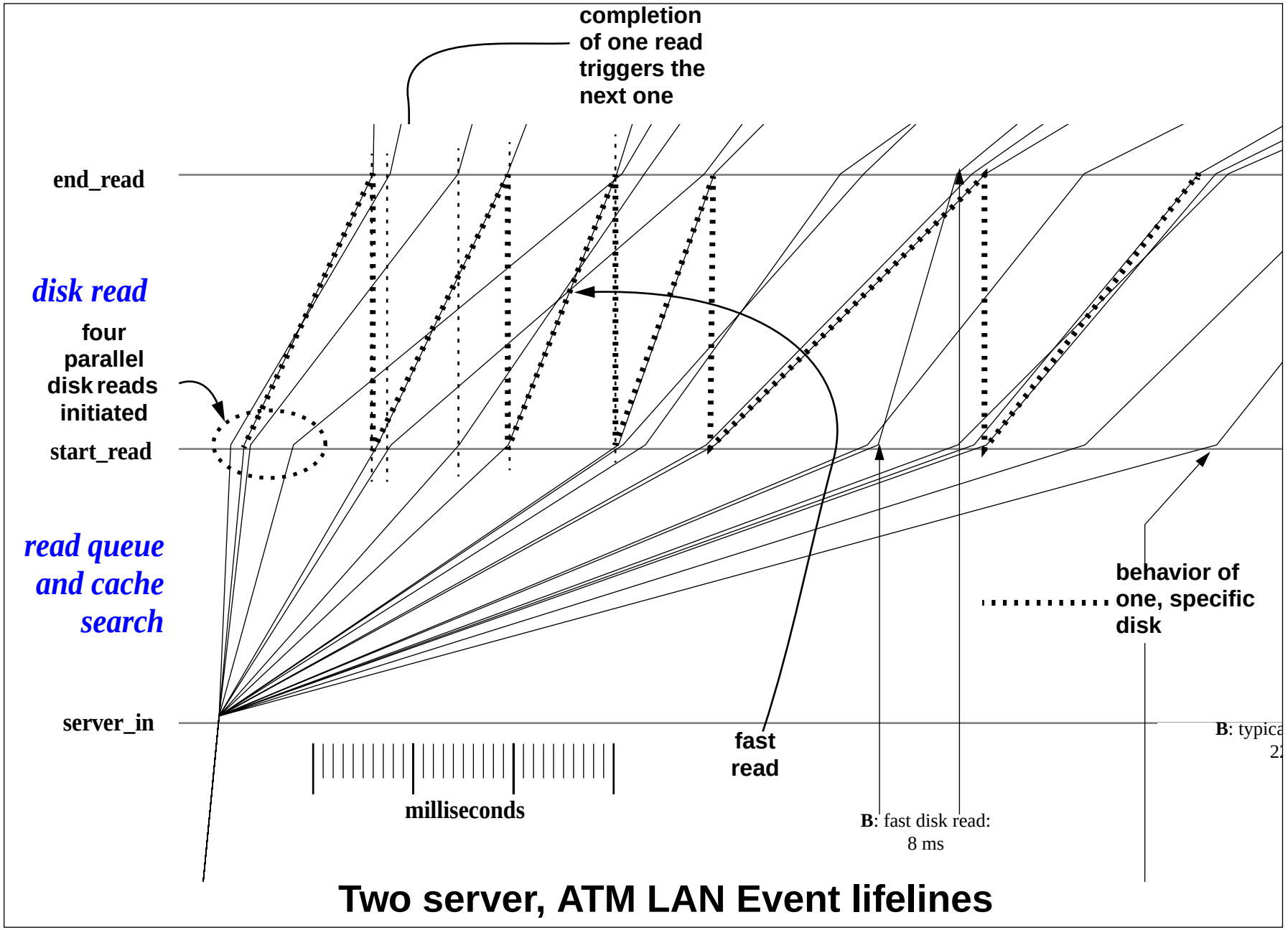
- **Analysis of the data block life-lines shows, e.g. (see next figure):**
  - **A: if two lines cross in the area between *start read* and *end read*, this indicates a read from one disk was faster than a read from another disk**
  - **B: all the disks are the same type, the variation in read times are due to differences in disk seek times**
  - **C: average time to move data from the memory cache into the network interface is 8.65 ms**
  - **D: the average time in disk read queue is 5 ms**
  - **E: the average read rate from four disks is 8 MBy/sec**
  - **F: the average send rate (receiver limited, in this case) is 38.5 Mb/sec.**
  - **G: some requested data are found in the cache (were read from disk, but not sent in previous cycle because arrival of new request list flushes write queue)**

NetLogger Analysis of the DPSS



Two server, ATM LAN

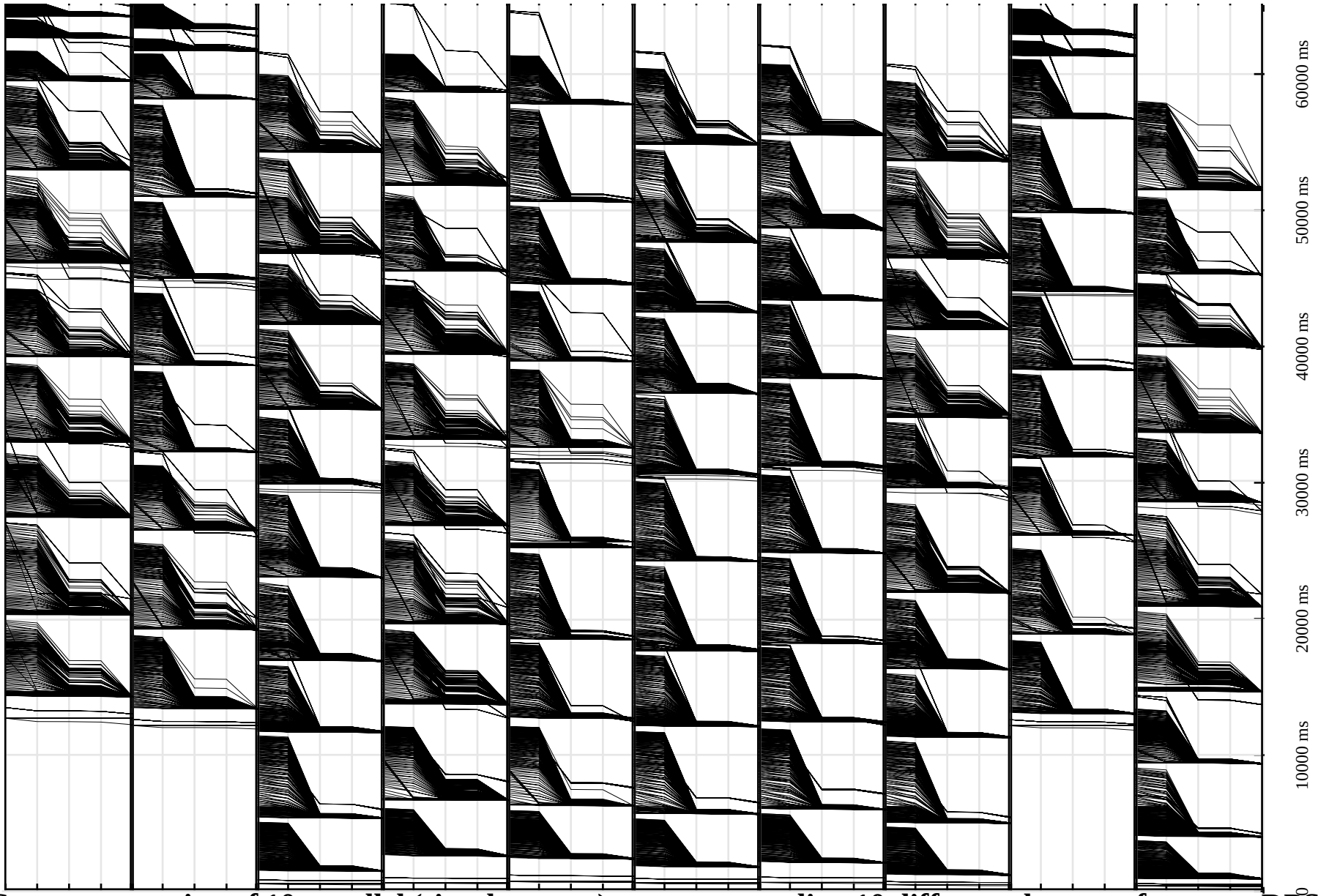
NetLogger Analysis of the DPSS



Other NetLogger Results: A DPSS scaling experiment:

- 10 clients accessing 10 different data sets simultaneously
- show that all clients get an equal share of the DPSS resources

NetLogger Analysis of the DPSS



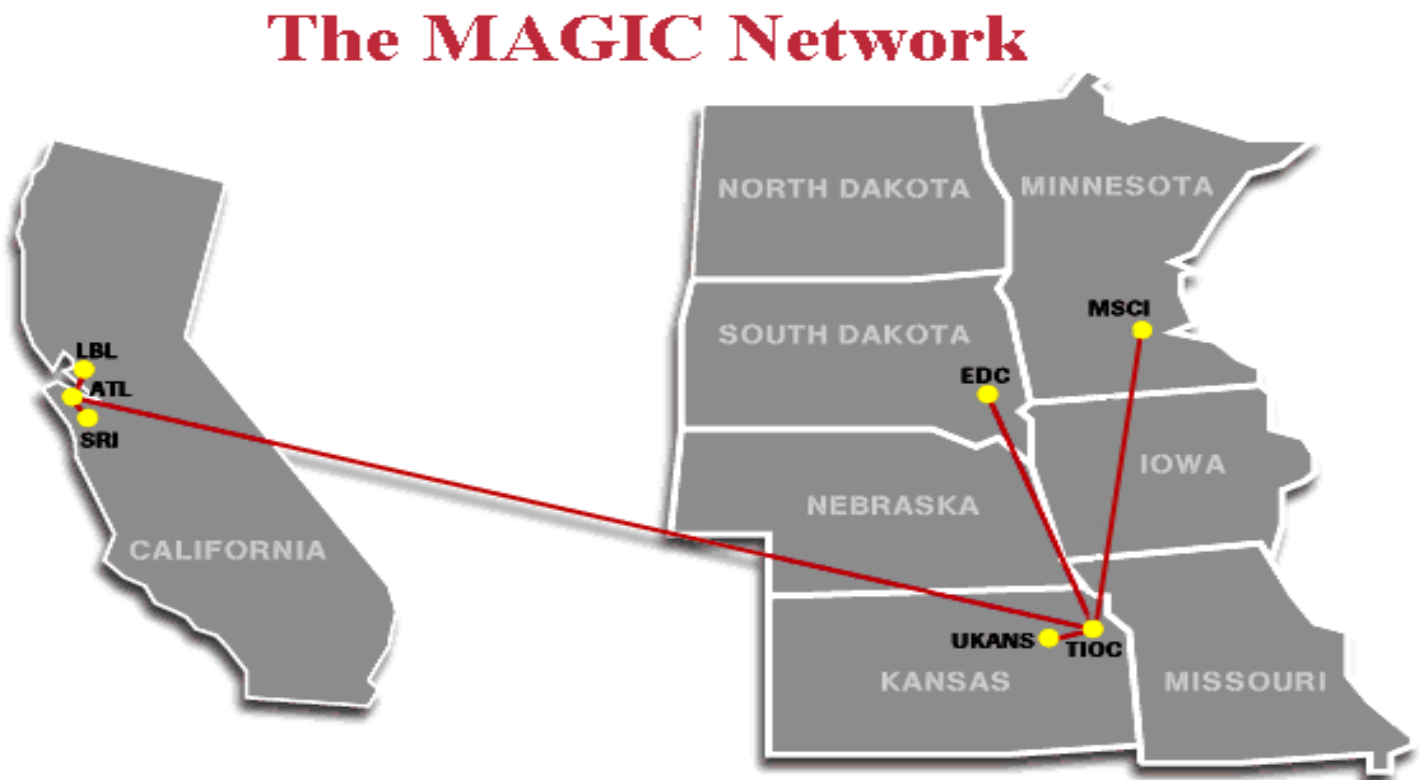
Correct operation of 10 parallel (simultaneous) processes reading 10 different data sets from one DPSS (each row is one process, each group is a request for 10 Mbytes of data, total = 1 GBy, *1.5 TBy/day*)

A Case Study: TCP in the Early MAGIC Testbed

- The Problem:
- DPSS and TerraVision were working well in a LAN environment, but failing to deliver high (even medium!) data rates to TerraVision when we operated in the MAGIC WAN.
- We suspected that the ATM switches were dropping cells, but they reported no cell loss.
- Network engineers claimed that the network was working “perfectly”.

MAGIC

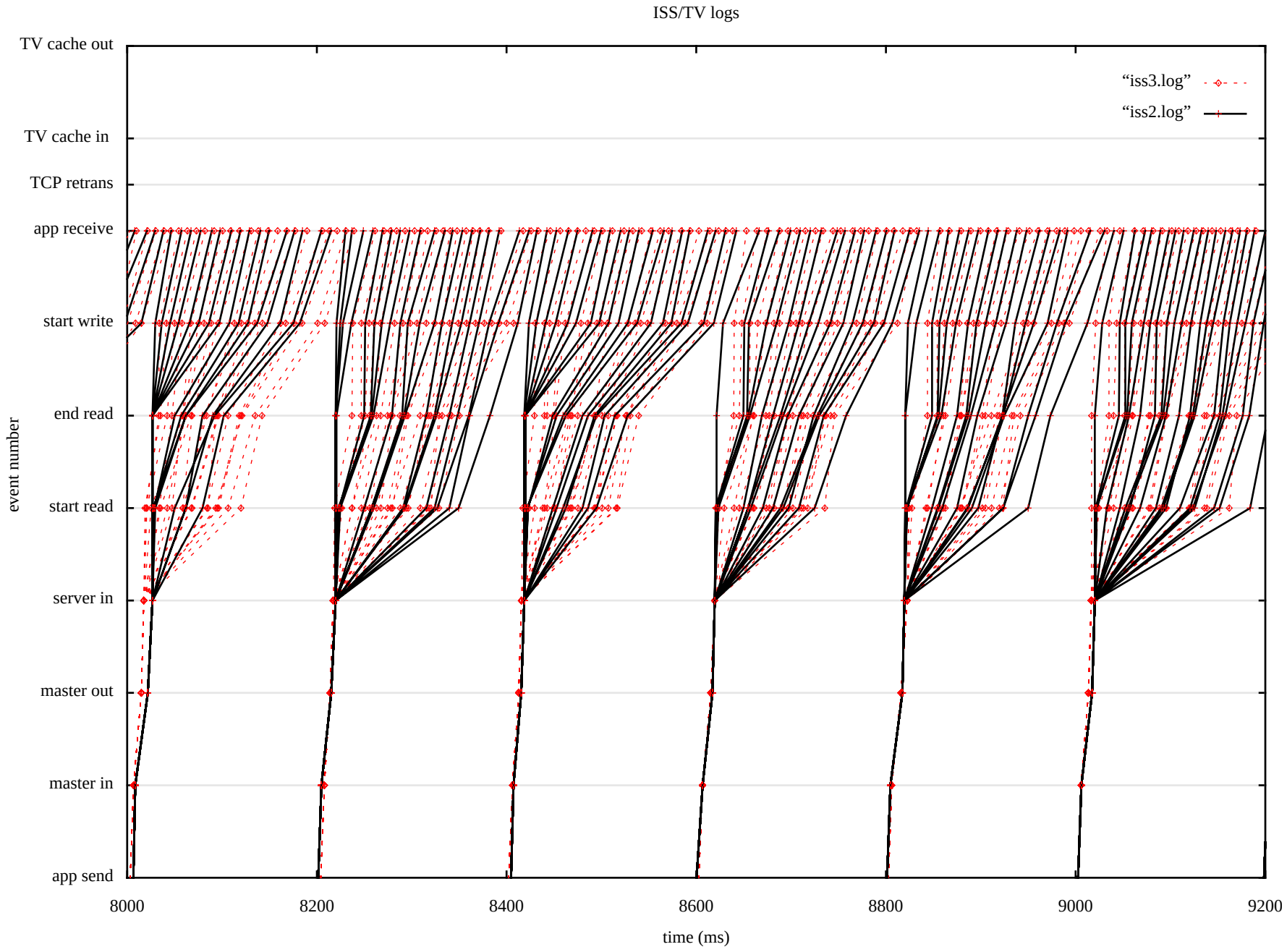
- **TerraVision image tiles are distributed across DPSS servers on the MAGIC network at the following sites:**
  - **EROS Data Center, Sioux Falls, SD;**
  - **Sprint, Kansas City, MO;**
  - **University of Kansas, Lawrence, KS;**
  - **SRI, Menlo Park, CA;**
  - **LBNL, Berkeley, CA**



Performance Analysis

- **Control test: 2 servers over ATM LAN**
  - **each line represents the history of a data block as it moves through the end-to-end path**
  - **data requests are sent from the application every 200 ms (the nearly vertical lines starting at *app_send* monitor point)**
  - **initial single lines fan out as the request lists are resolved into individual data blocks (*server_in*)**

Performance Analysis



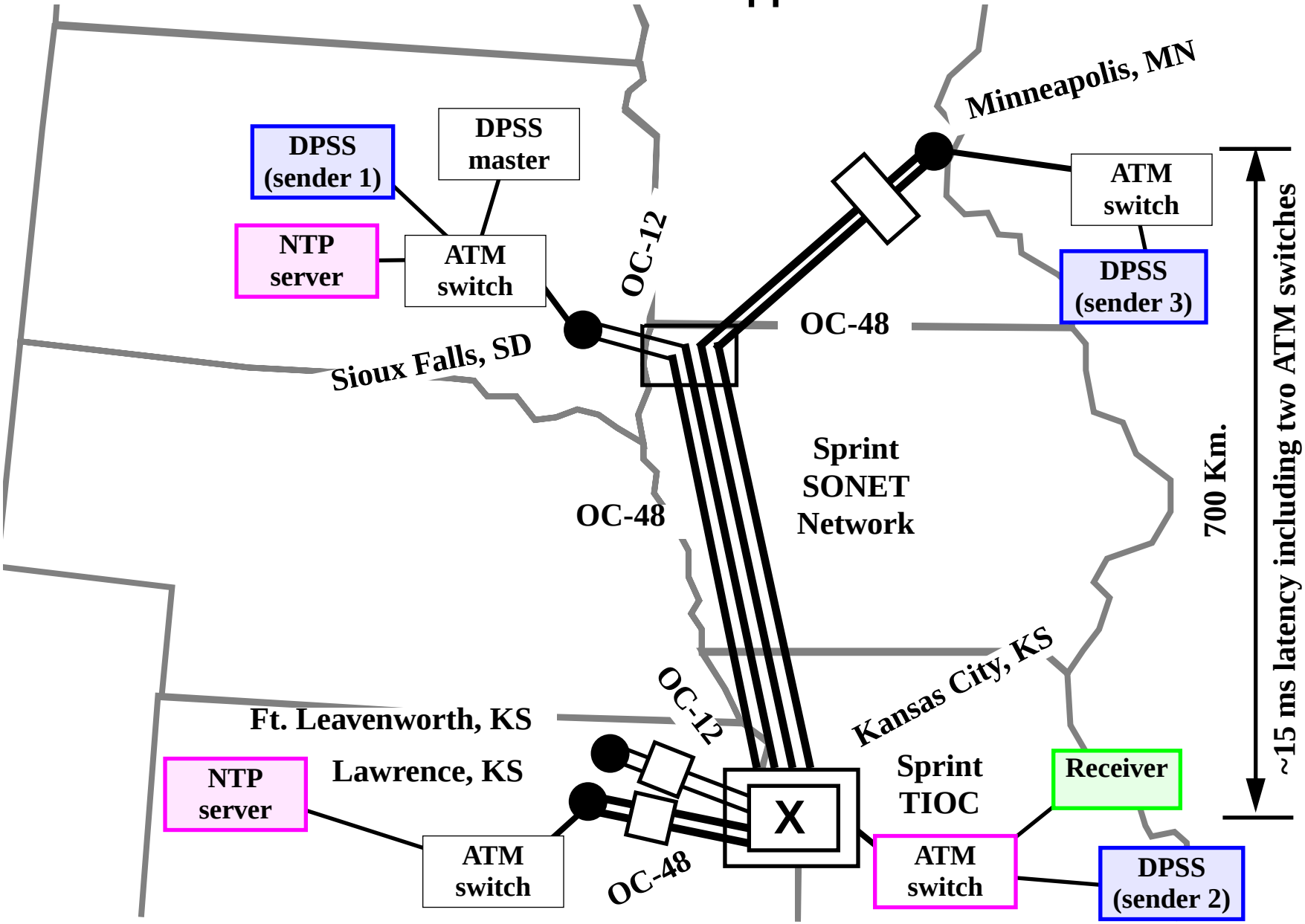
MAGIC WAN experiment:

- Three disk server configuration DPSS gave the results shown below:
- What we believe to be happening in this experiment is that TCP's normal ability to accommodate congestion is being defeated by an unreasonable network configuration:
  - the final ATM switch where the three server streams come together has a per port output buffer of only about 13K bytes
  - the network MTU (minimum transmission unit) is 9180 Bytes (as is typical for ATM networks)
  - the TCP congestion window cannot get smaller than the MTU, and therefore TCP's throttle-back strategy is pretty well defeated: On average, every retransmit fails, even at TCP's "lowest throughput" setting, because this smallest unit of data is still too large for the network buffers
  - three sets of 9 KBy IP packets are converging on a link with less than 50% that amount of buffering available, resulting in most of the packets (roughly 65%) being destroyed by cell loss at the switch output port

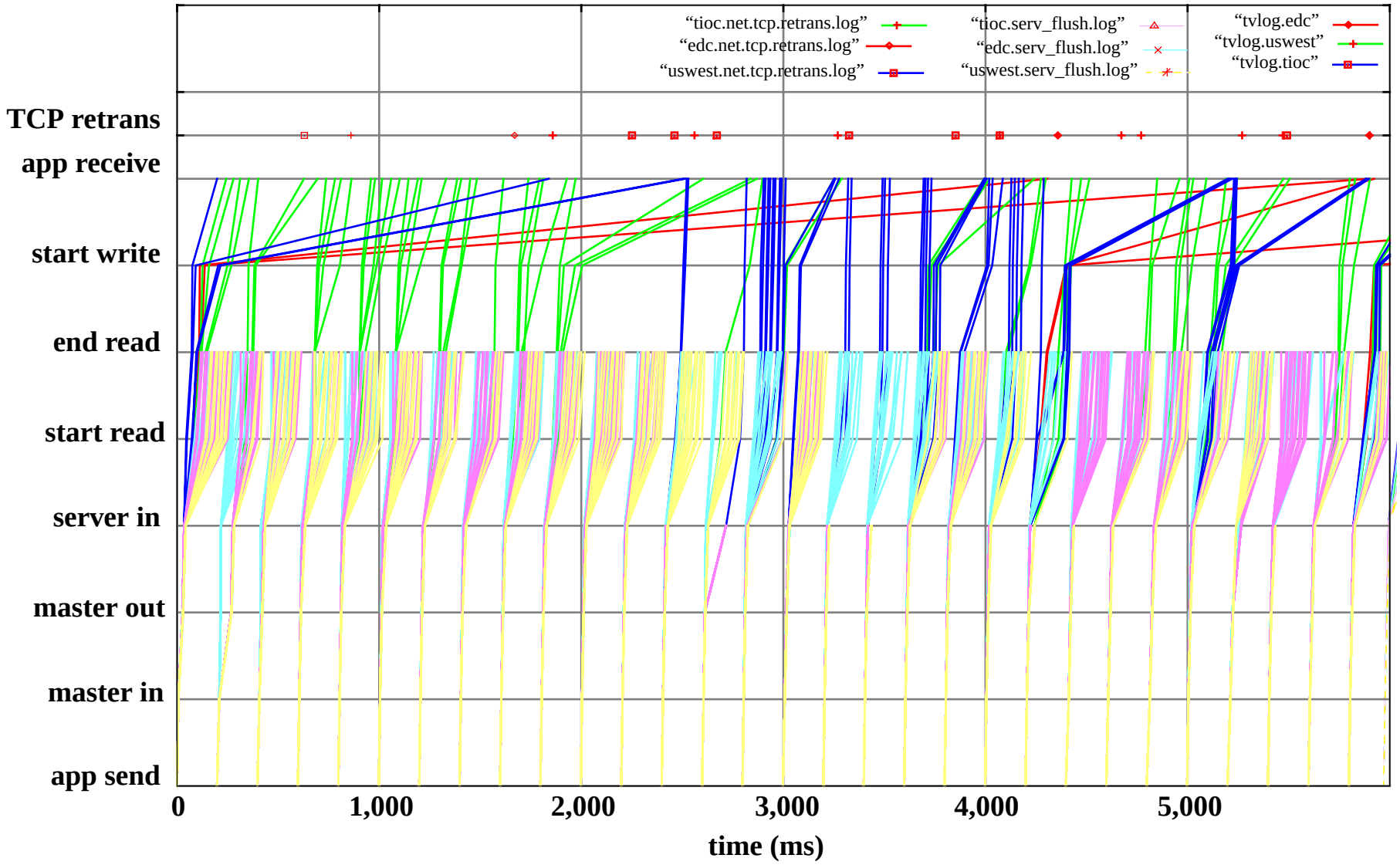
The new generation of ATM switches (e.g.: Fore LC switch modules) have much more buffering: 32K cells (1500 KB), so these switches should not have this problem.

Performance Analysis

The MAGIC Network and DPSS / Application Performance Test



Performance Analysis



Three servers, ATM WAN, SS-10s as servers, tv_sim on SGI Onyx

NetLogger Agents (Current work)

Management of monitoring programs and event logs for many clients connected to many distributed server components on many hosts is quite difficult.

Our approach:

- **use a collection of software agents to provide structured access to current and historical information**

Implementation:

- **written in Java**
- **uses the KQML communication language**
- **uses the Java Agent Toolkit (JATLite) from Stanford**
  - **JATLite: provides an agent framework, including basic communication tools and templates based upon TCP/IP and KQML messages**

NetLogger Management Agents

Agent Usage:

- monitor CPU load, interrupt rate, TCP retransmissions, TCP window size, etc.
- independently perform various administrative tasks, such as restarting servers or monitor processes
- Monitors when a server is being accessed, and triggers the agent monitoring of system info (e.g.: CPU load, interrupt rate, TCP retransmissions, TCP windows size) and server events. This reduces the amount of logging data collected.
- Collects event logs from each agent and merges them together, sorted by time, for use by the event log visualization tools.

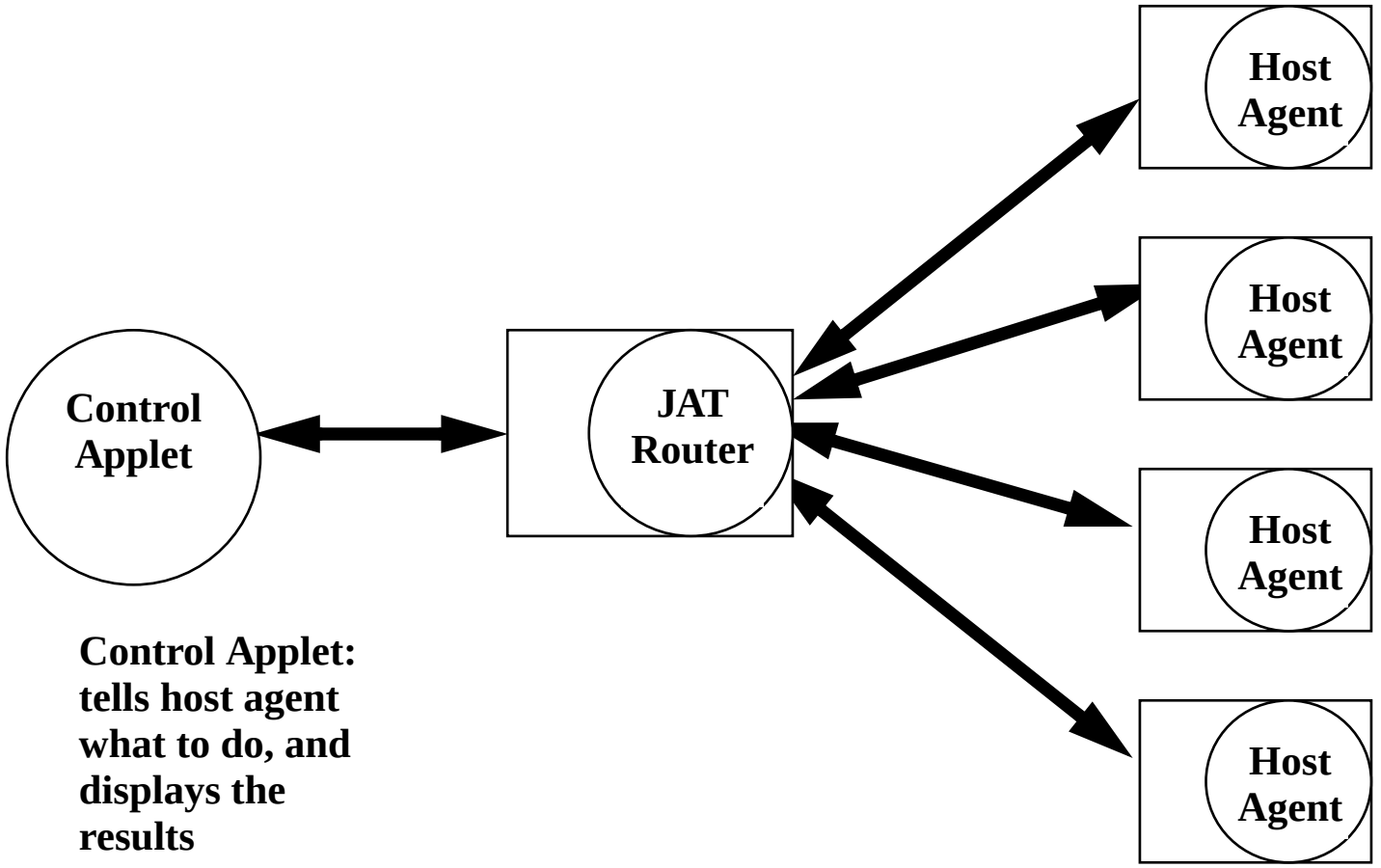
The agent architecture is a crucial component for NetLogger.

- without it management of the immense volume of log event data that can be generated would be infeasible

NetLogger Management Agents

- **NetLogger Agents:**
  - **Start/Stop Logging**
    - **vmstat and netstat**
  - **NTP status**
  - **Log Filtering**

NetLogger Management Agents



NetLogger Management Agents

screenshot of nl_vmstat/nl_netstat applet

NetLogger Management Agents

Status of All Magic hosts

art at the bottom to update at the given specific

l version of this applet

Hostname	Type	Clock Offset	NTP Hos
faraday.ukans.ma	MAGIC Master	0.001294	chronos.itt
faraday.ukans.ma	MAGIC Server	0.001294	chronos.itt
iss-4.lbl.magic.	MAGIC Server	0.000694	chronos1.lb
iss-3.lbl.magic.	MAGIC Server	***-0.014171***	tic.lbl.gov
iss-1-uni.sri.ma	MAGIC Server	0.003924	faraday.uka
kole.tioc.magic.	MAGIC Server	0.000662	blackstone-
blackstone.tioc.	MAGIC Server	0.000737	chronos.uka
edciss2.cr.usgs.	MAGIC Server	-0.000449	chronos-spa
edciss1.cr.usgs.	MAGIC Server	0.000799	chronos-spa

- **For more information see:**
  http://www-didc.lbl.gov/NetLogger