Xlans — A Graphical Display for Communication Protocols

Idit Keidar

High Availability Lab.
Computer Science department
The Hebrew University of Jerusalem
Jerusalem, Israel

Submitted as a Lab Project

November 11, 1992

1 Introduction

Xlans is a graphical tool for analysis of communication protocols, designed to analyze the behavior of *Transis* and the *ToTo* protocols. It depicts the flow of messages in the system and the relationships between them from a single processor's point of view.

Xlans comprises of two parts; one part is embedded in Transis, and extracts on-line information from it into a data file. The second part is an off-line graphical tool that displays the information from the data file. The graphical tool is independent of Transis, and may also be used to display and analyze other communication protocols.

The graphical interface of Xlans is based on a tool developed at University of California, Santa Barbara ([1]). Several adjustments and enhancements were made to match our needs. The differences are described in Section 5 below.

Xlans was implemented and is installed at the Hebrew University of Jerusalem. For Technical information on how to run Xlans see Appendix A.

2 Transis

In order to provide the necessary background to understand the graphical display, we include a brief description of Transis. A detailed description of the Transis environment and services can be found in [2].

Transis is a communication sub-system for high availability, developed at the Hebrew University of Jerusalem. The system comprises of a dynamic set of machines that communicate via asynchronous multicast messages. A multicast message leaves its source machine at once to all the machines in the system but may arrive at different times to them. Messages might be lost or delayed arbitrarily, but faults cannot alter messages' contents. Messages are uniquely identified through a pair $< sender, counter >$.

Each machine that executes Transis, supports a variety of reliable multicast message passing services. Messages are sent onto the network and received into a common data structure, called the *causal DAG* (*Directed Acyclic Graph*). All the services are provided by delivering messages that reside in the DAG.

The DAG is a representation of the *causal order* defined below: the nodes are the messages, the arcs connect two messages that are directly dependent in the causal order[1]. The causal graph contains all the messages sent in the system. The processors see the same DAG, although as they progress, it may be "revealed" to them gradually in a different order.

Causal Multicast

We define the *causal order* of messages, motivated by Lamport's definition of order of events in a distributed system ([4]), as the reflexive, transitive closure of:

(1) $m \stackrel{cause}{\longrightarrow} m'$ if $\text{receive}_q(m) \rightarrow \text{send}_q(m')$ [2]

(2) $m \stackrel{cause}{\longrightarrow} m'$ if $\text{send}_q(m) \rightarrow \text{send}_q(m')$

[1] An arc from A to B means that B acknowledges A. Therefore A "generated" B.

[2] Note that '$\rightarrow$' orders events occurring at q sequentially, and therefore the order between them is well defined.

If m' follows m in the causal order, we say that m' *follows* m. Message m is *prior to* m' if m' follows m. Messages m, m' are *concurrent* if m does not follow m', and m' does not follow m. The set of all messages concurrent to m is called the *concurrency set* of m.

The Transis system guarantees the causal delivery order of messages. In Transis, each newly emitted message contains piggybacked ACKs (acknowledgments) to previous messages. The ACKs form the $\xrightarrow{cause}$ relation directly, such that if m' contains an ACK to m, then $m \xrightarrow{cause} m'$. Implicit ACKs are readily available using the transitive closure of the $\xrightarrow{cause}$ relation. If a message arrives at a machine, and some of its causal predecessors are missing, Transis transparently handles message recovery and re-ordering.

Transis Services

The operation of a multicast service in the Transis environment is primarily associated with two *interface events*: the service *receives* messages from the DAG, delays them according to various system conditions and then *delivers* them to an upper level.

Transis provides a variety of reliable multicast services. The services use different delivery criteria on the messages in the DAG. Here is the list of services:

1. **Atomic** broadcast: guarantees delivery of the message at all the active sites. This atom delivers the message immediately to the upper level.
2. **Causal**: guarantees that if $m \xrightarrow{cause} m'$ as defined above, then for each processor p,

   $$\text{delivery}_p(m) \rightarrow \text{delivery}_p(m').$$

   This is implemented by delaying delivery until all acknowledged messages have been delivered.
3. **Agreed**: delivers messages in the same order at all sites.
4. **Safe**: delivers a message when all the processors have acknowledged its reception.

The user can use any type of multicast service for each message. Messages are held by Transis for retransmission as long as they may be demanded.

ToTo

The *ToTo* protocols are a family of protocols for total ordering of messages in broadcast domains. The ToTo protocols implement the "agreed-multicast service" in *Transis*. For details see [3].

Index of Synchrony

Let m be a multicast message sent by machine p. We define the *index of synchrony* of m as the number of machines p must receive messages from, (including p,) before it can deliver m to the upper level. These messages may be concurrent to m, or follow m. The index of synchrony depends on delivery criteria:

- The efficiency of an **agreed** multicast protocol may be measured in terms of the worst case and average case index of synchrony it incurs on agreed multicast messages.

- For **causal** messages the index of synchrony is always 1.
- For **safe** messages it is always n, the number of processors in the system.

3 The Graphical Tool

The graphical package is implemented using the Motif library, on the X window graphics platform. It is portable, and may be run independently of Transis.

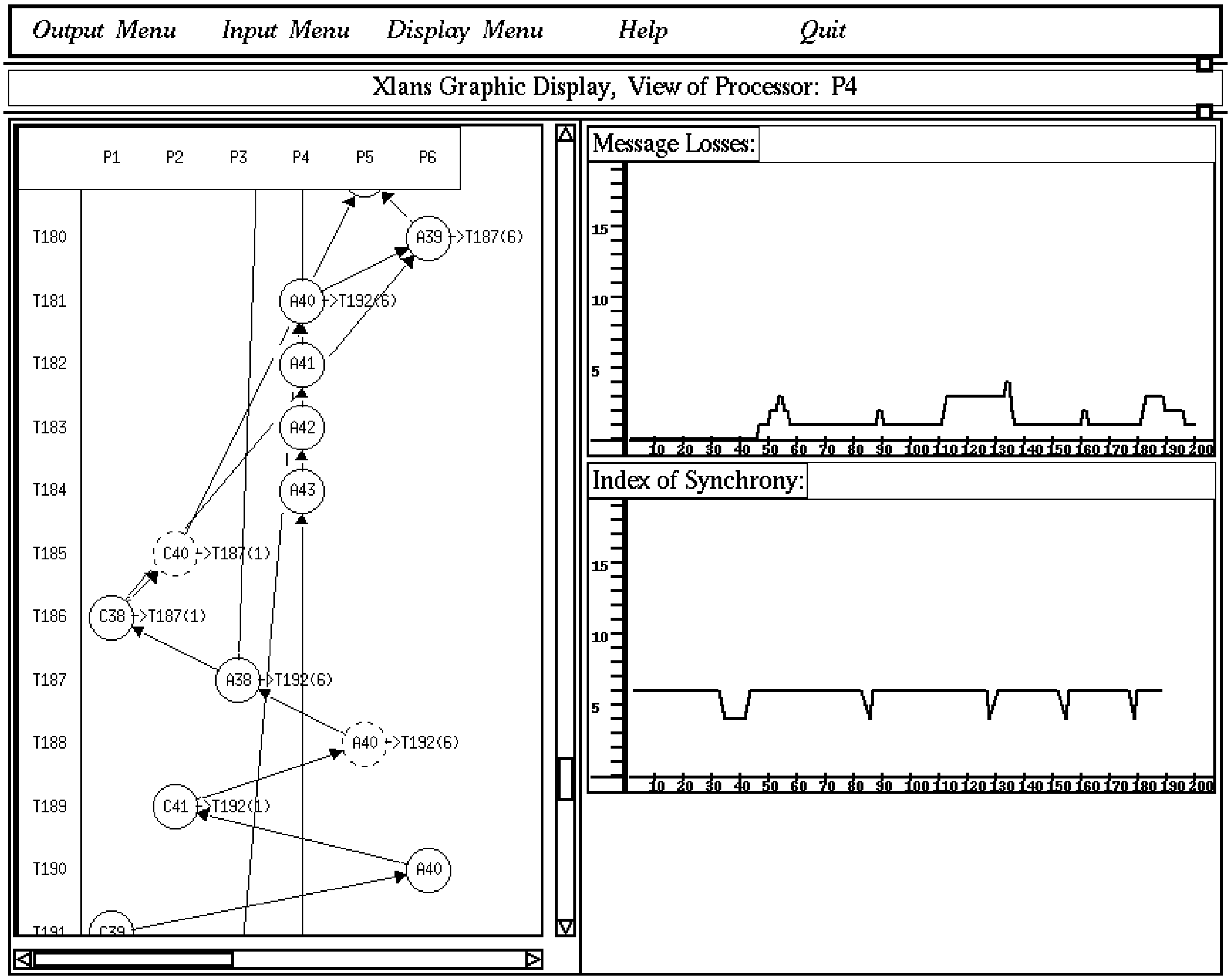

Figure 1: Snapshot of the graphical tool

The user interface is controlled via pull-down menus for the selection of input, output and display parameters. The display is roughly divided into two parts: a graphical representation of the message flow in the system, and a graph section showing various Transis parameters. A snapshot of the graphical display is shown in Figure 1.

3.1 Menu Interface

The user interface is controlled via four pull-down menus.

- Using the **input menu**, the user may select a data file to display information from.

- Using the **output menu**, the user may print information about the message flow in text format or in LaTeX format into a file, or prepare a file with graphic output to the printer - in postscript.
- The **display menu** enables the user to choose among various display options, e.g. whether to show delivery timestamps.
- The **help menu** supplies on-line help for the graphical display.

3.2 Graphical Representation of Message Flow

The message flow, including acknowledgments, is shown from the point of view of a single processor. The viewing processor's number is displayed in the title of Xlans.

The causal DAG of messages is displayed (see Section 2 above). Nodes are messages, and the directed arcs represent the causal order between them. Xlans assigns *timestamps* to messages. The timestamps represent the order of arrival. The order never violates the causal order; any internal recovery/retransmission of messages is done transparently by Transis, and Xlans is oblivious to it.

The display is arranged as a two dimensional grid, the rows represent Xlans timestamps, so that one message is shown in each row. The columns represent processors. Each message appears in the column of the processor that sent it. A message is represented by a circle labeled with the following information: A letter representing its *Transis type*, which determines the delivery criteria for the message, e.g. C for Causal, A for Agreed. And a counter (sequence number) generated by the sending processor. Solid circles represent original transmissions and dashed circles — retransmissions.

The tool may be used to display large numbers of messages and processor interactions, and the user can scroll around to investigate areas of interest.

Example

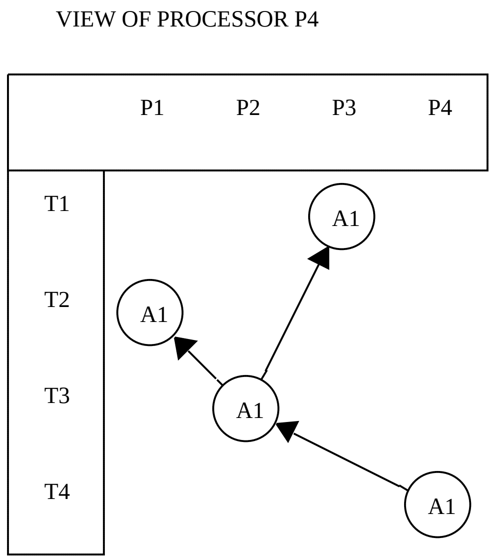

Figure 2: Example of message flow display

In Figure 2 we see an example of the message flow display, with 4 processors.

Time 1: Processor 3 sends an agreed message, this is its first message so its sequence number is 1. We will refer to this message as $m_{3,1}$.

Time 2: Processor 1 sends an agreed message, $m_{1,1}$.

Time 3: Processor 2 sends an agreed message, acknowledging messages $m_{1,1}$ and $m_{3,1}$.

Time 4: Processor 4 sends an agreed message, explicitly acknowledging message $m_{2,1}$ and thus implicitly acknowledging having received the messages acknowledged by it: $m_{1,1}$ and $m_{3,1}$.

3.3 Delivery Information

An important aspect of the Transis and ToTo Protocols, is the criteria for delivering messages to the upper level. In order to analyze these protocols, it is helpful to trace the messages' delivery time. To this end, we supply as an option the display of delivery timestamps.

The delivery timestamp of a message is the Xlans timestamp of the last message received in the system before the delivery. Notice that several messages might be delivered at the same timestamp, since receiving one message may fulfill delivery criteria for several others.

When this option is requested, Xlans displays the timestamp at which the message was delivered beside each message, and along with it, the index of synchrony (defined in Section 2 above) of the delivered message appears in parentheses.

Example

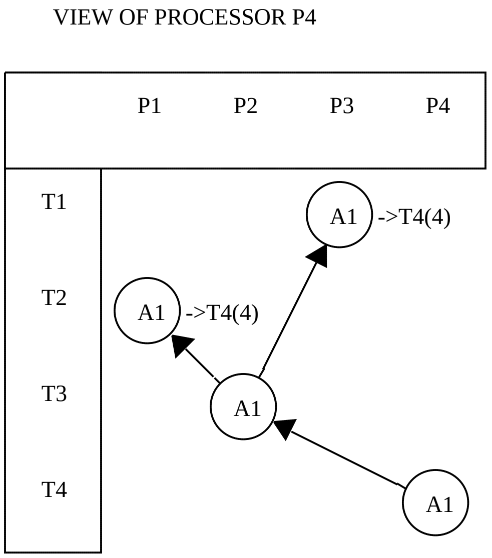

Figure 3: Example of message flow display with delivery timestamps

In Figure 3 we see a display of the message flow depicted in Figure 2, with delivery timestamps shown. At timestamp 4, after having received one message from each of the participating processors, the first two messages are delivered to the upper level. The messages $m_{3,1}$ and $m_{1,1}$ are concurrent messages in the causal order, both messages are delivered at this timestamp, and their index of synchrony is 4, since messages from all 4 processors were received before the protocol decided to deliver them.

3.4 Graphs

The graph section of the display shows statistical information gathered from Transis, against time. The horizontal axis of the graphs corresponds to the last valid Xlans timestamp. Two graphs are shown:

- A graph depicting **message losses**: the number of messages the processor knows it is missing. If a message is lost, the processor discovers that it is missing when it sees other messages acknowledging it. When the message is received from retransmission, it ceases to be missing. The number of lost messages is collected by Xlans every time a message is received in Transis.

- A graph showing the **index of synchrony** at delivery time of **agreed** messages against their sending timestamps. If concurrent messages are delivered at the same timestamp, they have the same index of synchrony, calculated upon the decision to deliver the entire concurrency set.

  Only messages with agreed delivery criteria participate in this graph, which is used to analyze ToTo.

3.5 Example

In Figure 4 we show a larger example, from an actual run of Transis. The viewing processor purposely lost 5 percent of the messages. The figure and the accompanying text were generated by Xlans output menu options.

Example of message flow with 6 processors, from the viewpoint of processor 4.

Time 1: Processor 6 sends an agreed message, $m_{6,39}$.

Time 2: Processor 4 sends an agreed message, $m_{4,40}$, acknowledging $m_{6,39}$.

Time 3: Processor 4 sends an agreed message, $m_{4,41}$, acknowledging $m_{4,40}$.

Time 4: Processor 4 sends an agreed message, $m_{4,42}$, acknowledging $m_{4,41}$.

Time 5: Processor 4 sends an agreed message, $m_{4,43}$, acknowledging $m_{4,42}$.

Time 6: Processor 2 sends a causal message, $m_{2,40}$. The message is received from retransmission.

Time 7: Processor 1 sends a causal message, $m_{1,38}$, acknowledging $m_{6,39}$ and $m_{2,40}$.

Time 8: Processor 3 sends an agreed message, $m_{3,38}$, acknowledging $m_{1,38}$. Delivery of $m_{6,39}$, $m_{2,40}$ and $m_{1,38}$, the index of synchrony is 6.

Time 9: Processor 5 sends an agreed message, $m_{5,40}$, acknowledging $m_{3,38}$. The message is received from retransmission.

Time 10: Processor 2 sends a causal message, $m_{2,41}$, acknowledging $m_{5,40}$.

Time 11: Processor 6 sends an agreed message, $m_{6,40}$, acknowledging $m_{2,41}$.

Time 12: Processor 1 sends a causal message, $m_{1,39}$, acknowledging $m_{6,40}$.

Time 13: Processor 3 sends an agreed message, $m_{3,39}$, acknowledging $m_{4,40}$ and $m_{1,39}$. Delivery of $m_{3,38}$, $m_{4,40}$, $m_{5,40}$ and $m_{2,41}$, the index of synchrony is 6.

Time 14: Processor 4 sends an agreed message, $m_{4,44}$, acknowledging $m_{3,39}$ and $m_{4,43}$.

Time 15: Processor 4 sends an agreed message, $m_{4,45}$, acknowledging $m_{4,44}$.

Time 16: Processor 5 sends an agreed message, $m_{5,41}$, acknowledging $m_{3,39}$. The message is received from retransmission.

Time 17: Processor 2 sends a causal message, $m_{2,42}$, acknowledging $m_{5,41}$.

Time 18: Processor 6 sends an agreed message, $m_{6,41}$, acknowledging $m_{2,42}$.

Time 19: Processor 1 sends a causal message, $m_{1,40}$, acknowledging $m_{6,41}$.

Time 20: Processor 3 sends an agreed message, $m_{3,40}$, acknowledging $m_{4,41}$ and $m_{1,40}$.

Time 21: Processor 5 sends an agreed message, $m_{5,42}$, acknowledging $m_{3,40}$.

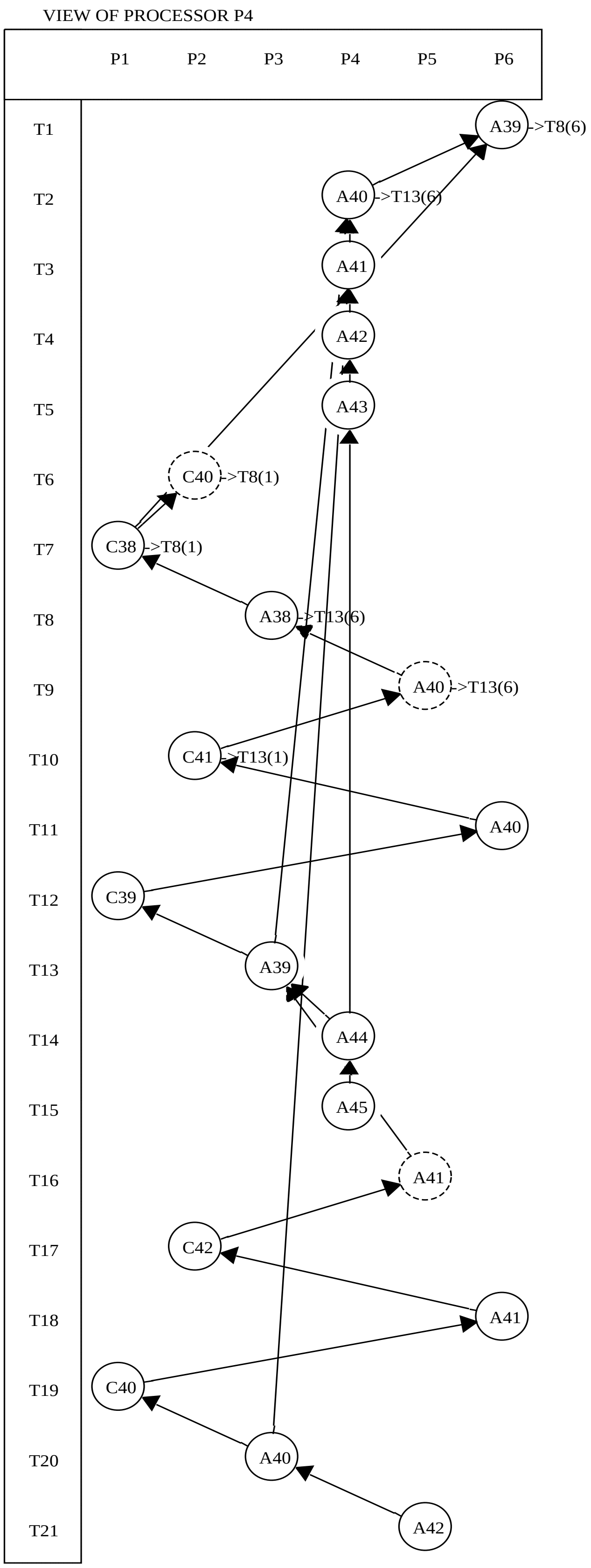

Figure 4: Example of message flow in Transis

4 Extracting Data from Transis

In Transis, messages are received and kept in internal data structures. When delivery criteria for a given message are fulfilled, Transis delivers the message to the upper level. If information gathering is requested, Xlans collects data about the messages during Transis execution into a file.

Information about a message is collected at two points during its life time:

- Upon insertion into the causal DAG of messages, general information about the message is collected. The message id, consisting of the number of the sending processor and a counter, generated by it. The Transis type. Whether this is original transmission or retransmission. And acknowledgments piggybacked on this message. Also, the number of messages known to be missing is collected.
- When delivered to the upper level, its index of synchrony is collected.

The user may request to gather information about the flow of messages in the system, and specify the number of messages to gather information about. Once the requested amount of messages was received in Transis, and the information about them was extracted, the data file is closed, and is ready to be read by the graphical tool.

There are two ways to invoke information gathering:

- When running Transis the user may specify that he wishes to gather information as a command line parameter. He may specify a starting message number and the amount of messages. The starting message is important in order to analyze system behavior after it has been running for a while, and not necessarily at the beginning.
- It is also possible to issue on-line requests to Transis via a monitor. Thus, the system supports an unlimited number of requests in a single session of Transis, each creating a separate data file, depicting the system's behavior at a different time interval. The monitor for Transis is currently being implemented.

5 Differences between Xlans and UCSB graphical tool

Unlike Xlans, UCSB's tool gathers information from a simulator of Trans, and not from an actual implementation. The simulator has a global view of the system that is not available to a single processor. Xlans shows local information of a single processor. The most significant difference between the local and global views is the concept of lost messages. In the global view, both the original transmission of a lost message and its retransmission are shown. The retransmission is shown at a later timestamp, out of causal order. A single processor, however, is only interested in each message once. If it did not lose it, it will ignore the retransmission, and if it did lose it, it will solely see the retransmission. The retransmitted message will be placed where the original message belongs in the causal order. Each processor will eventually see all messages flowing in the system. Xlans shows the global message flow as it is revealed at a given node of Transis, and helps analyze the decisions of the delivery algorithms at that node.

Another feature of Xlans is the independence of the graphical tool. The graphical tool reads data from a file, and may be run independently of Transis. Thus, Xlans is more flexible, and can easily be used to analyze communication protocols other than Transis and ToTo.

Xlans is implemented in Motif.

New Features In Xlans

There are several additions in Xlans.

- Xlans includes an optional display of delivery timestamps and index of synchrony, to help analyze delivery criteria of the protocols.
- It contains graphs of Transis parameters that enhance understanding of system behavior over a long period of time, and the effects of message losses on system behavior.
- Another important feature is the ability to gather information in the middle of a Transis session, and to do so an arbitrary number of times during one session.
- We also supply on-line help for Xlans.

6 Future Directions

Further developments planned for the future include:

- The display of membership changes during the message flow, in order to follow the membership algorithms in Transis.
- Display of latency against time, along with existing graphs.
- On-line invocation of Xlans information gathering via a monitor.

References

[1] D. A. Agarwal and L. E. Moser. A graphical interface for analysis of communication protocols. In *Proceedings of the 20th ACM Computer Science Conference*, pages 149–156, Kansas City, MO, March 1992.

[2] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A communication sub-system for high availability. In *FTCS conference*, number 22, July 1992. previous version available as TR CS91-13, Dept. of Comp. Sci., the Hebrew University of Jerusalem.

[3] D. Dolev, S. Kramer, and D. Malki. Total ordering of messages in broadcast domains. Technical Report CS92-9, Dept. of Comp. Sci., the Hebrew University of Jerusalem, 1992.

[4] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 78.

A Running Xlans

Xlans is installed at the High Availability Lab. at the Computer Science department, Hebrew University of Jerusalem. The programs may be found in the directory */ha/src/xlans*. A *README* file in this directory explains how to run Xlans:

```
To run the graphical tool type:
        run_xlans [ data file name ]
Use the input menu to select a data file, using a file selection box.

To see a demo of Xlans, you may use the demo data files
in this directory.

To create a data file for Xlans run transis with command line
parameters -g<number of messages to gather data about>,
           -gs<number of messages to wait until gathering data>.

Example:
transis -g50 -gs200 config

Will run Transis, and wait during the passing of 200 messages
in the system, and then will gather information about
the next 50 messages passed in the system.

A data file will be created, with a name
consisting of the host_name and process_id.

Example:
If transis -g was run on Hal, and the process number was 14179,
the resulting data file name is: hal_14179_1.stat.

For further details see on-line help of Xlans.

For documentation of the structure of the data file see data.doc.
```

The README file also contains brief descriptions of the involved files.