XAV: A Tracing Framework for Exploring Large Network Simulation Outputs

Ryad Ben-El-Kezadri
LIP6, Université Paris VI,
104 avenue du Président Kennedy
75016 Paris - France
ryad.bek@lip6.fr

Guy Pujolle
LIP6, Université Paris VI,
104 avenue du Président Kennedy
75016 Paris - France
guy.pujolle@lip6.fr

Farouk Kamoun
CRISTAL, Université de la Manouba,
Campus Manouba, 2010 - Tunisia
frk.kamoun@planet.tn

ABSTRACT

This paper presents our ongoing works towards visual exploration of large network simulation outputs. Visual exploration allows users to search through simulation traces by using multi-dimensional representations of the network in an intuitive and interactive way. To speed up trace file exploration, we propose to store the simulation events according a format, namely XAV, that i) exploits the multidimensional nature of users requests and ii) allows quick identification of the packet paths through the network. XAV files are stored in a database to simplify data manipulation. The XAV tracing framework has been implemented in the NS-2 simulator and tested over a wireless ad-hoc network composed of 25 nodes. The performance evaluation shows that XAV enables to extract about 1000 packet paths per second from 100 MB trace files.

Categories and Subject Descriptors

I.6 [**Simulation and modeling**]: Simulation Output Analysis; H.2.1 [**Database Management**]: Logical Design – *Data models*; H.2.8 [**Database Management**]: Database Applications – *Statistical databases.*

General Terms

Standardization, Design.

Keywords

Network simulation, Exploration, Database, Data path, Trace file, XML, XQuery.

1. INTRODUCTION

Networks are becoming increasingly large, dynamic and complex. In this demanding environment, the simulation tools are not only expected to faithfully simulate networks, but also to enable users to understand the whys and hows of the obtained simulation results. The larger and more complex the networks, the larger the simulation traces are and the more users will need freedom to explore the simulation outputs. Data exploration goes beyond traditional data exploitation as it allows the user to navigate and interact intuitively with the outputs. Contrary to classic data exploitation which limits the analysis to general and surface examinations, data exploration enable the users to enter in the trace files and to access every piece of information they contain.

Data exploration relies on extensive trace file processing. Trace processing is a complex task because the useful information can be deeply buried into the files. Two processing methods can be distinguished at this level. In the traditional method, a procedural language is used for filtering and aggregating the relevant trace records. Huginn [1] as the majority of post processing tools relies on this principle. Huginn allows the visualization and interaction with NS-2 wireless traces at the MAC level. The Huginn evaluation engine reads the trace in sequential order, correlates the events of interest and finally generates an intermediate network representation with which the user can interact. The problem of this approach is that the filtering and the correlation of the send events with their corresponding receive events on the MAC layer are complex, memory-expensive, and involve a lot of code. The second method uses non procedural languages. Non procedural query languages such as SQL or XQuery do not require writing traditional programming logic. Users only concentrate on defining the inputs and outputs rather than the programming steps required in a procedural language. Non procedural languages also have the advantage to work with databases which provide built-in indexing facilities to speed up access to data. Non procedural languages relieve applications from complex memory management as most underlying functions are delegated to an external processor. To our best knowledge, JTrana [2] is the only tool that uses a database management system (DBMS) to process the simulation outputs. JTrana displays general information on the network, nodes remaining energy level and packet statistics. Different tables are used for packet events, nodes movements and energy but each table row roughly corresponds to an event record. Such use of DBMSs is a clear step towards innovative applications. However, a model that captures the fine structure of the simulation outputs has not yet been defined for real time exploration. Indeed, JTrana does not perform well with large trace files [3] and has not been designed for visual exploration.

Data exploration will rely on new paradigms. Until now, for reasons of simplicity, the post processing applications have been developed separately, each one operating on potentially different simulation outputs. For example, the popular NS-2 simulator [4] generates NAM files for specific animation purpose and TR traces for packet and queue statistics analysis. Specific formats have also been released to provide statistics on network components and flows [5]. In future systems, the standalone applications will be replaced by an interconnected set of services. At the center of the system, the exploration service will act as a portal projecting representations of the network from which users can interact, navigate and launch other services.

The XML tracing framework of Yavista (XAV) has been built to support the various requirements involved in these tasks. Yavista is a post processing toolset dedicated to wireless simulation exploration [6], [7]. Yavista directly instruments the simulators in neutral language. Yavista neutral language is a simple, highly expressive, and well defined declarative language dedicated to network simulation. It operates as an interface between the post processing layer and the source layer. From this interface, simulators appear as neutral code generators and only distinguish themselves by non functional aspects such as performance, completeness (number of protocols implemented) and ease of use.

XAV and Yavista conceptual architecture is described in figure 1.

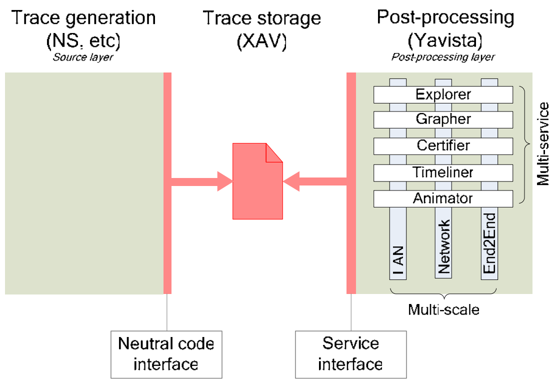


Figure 1. Yavista/XAV conceptual architecture.

The XAV framework relies on the XML standard [8] and XML enabled databases. XML enabled databases are extensions of RDBMS which transparently map the XML documents into their own data structures. Users access the XML data through a non-procedural query language called XQuery [9]. XQuery acts much like SQL. It has been designed to select the XML data elements of interest, reorganize them and return the results in the desired form.

XAV final objectives are the followings:

i. *Allow interaction with simulation outputs*: an important feature of XAV is to define the dimensions along which the user can interact (protocols, time and space) and to model them as first class items in the trace files. In comparison, flat trace files only expose one implicit entry point, this latter being the first line of the file and having no type.

ii. *Allow data navigation*: the second important feature of XAV is to be data-path oriented: under XAV, the complete path of a packet in the network can be retrieved from an ID. As a result, XAV outputs are not considered as independent data records but as collections of data paths. Furthermore, XAV data lie in visible contexts and these contexts can be used by the processing tools to find the relationships between the packet paths.

iii. *Seamless integration into simulators*: The integration of XAV does not require modifications in the protocol of simulators. Only the tracing module changes. XAV actually adds two minimal preformatting stages. The slowest, i.e. the copy of XAV outputs in the database, is done at 200MB/mn through a single native DBMS command.

iv. *Simplify post processing tools design*: Under XAV, the memory management and the query processing are handled by external components. Memory management which is critical, especially when processing very large amount of data, is moved from the applications to the database and the hard coded filtering is replaced by XQuery commands. Complex intermediate representations are no more needed because the returned results are formatted on-demand directly from the simulation outputs.

v. *Overcome the 100 MB and 1 GB trace size limits:* The minimum time to parse a flat file of a hundred of MB with an AWK script[1] is about ten seconds. As complex queries often need to scan documents multiple times (to retrieve preliminary information) the response time may not be tolerable for users. XAV outperforms traditional systems for targeted queries, that is to say queries that retrieve a small set of closely related tuples. The efficiency of these queries is especially important because they are extensively used for data exploration.

XAV architecture and performance for traditional service delivery have been discussed in [3]. This paper covers the interaction and navigation processes.

The rest of this paper is organized as follows. Chapter 2 describes the practical implementation of XAV in NS-2. Chapter 3 focuses on navigation and data path extraction. Chapter 4 details the interaction aspects between XAV and users. Chapter 5 evaluates the performance of XAV under three test scenarios. Chapter 6 concludes the article and presents future works.

2. IMPLEMENTATION OF XAV UNDER NS-2

XAV logs events as all tracing systems. XAV actually introduces two changes to conventional systems:

1. Under XAV, packets data and send and receive nodes operations are not stored together but in two distinct files. The first file, namely the data file, lists the packets payloads and headers generated during simulation with their different fields: source, destination, type, length, etc. The second file, namely the operation file, lists the nodes operations.

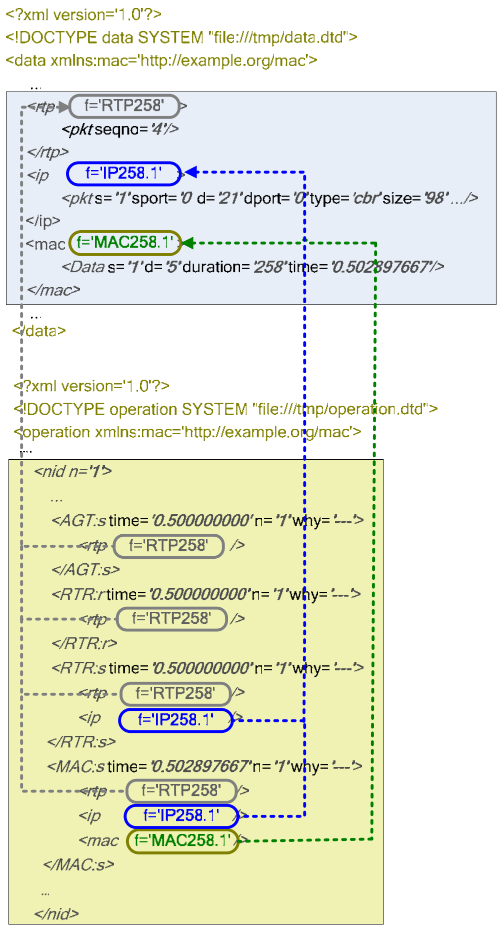
2. The nodes send and receive operations are not enumerated in the same order as they are generated but are grouped according their node id.

A basic XAV output is displayed in figure 2. The top of the figure shows an excerpt from a data file. One RTP, one IP and one MAC header are represented. Each header is identified by a unique ID.

[1] AWK and Perl are the most commonly used tools to process trace files.

The RTP, IP and MAC headers respective IDs are *RTP258*, *IP258.1* and *MAC258.1*. Each header carries its own information. For instance, the MAC header refers to a data frame (element *data*) and this frame is sent by node 1 (attribute *s*) to node 5 (attribute *d*).

The lower half of figure 2 is extracted from an operation file. It shows the operations carried out by node 1 (attribute *n* of *nid*) to send an RTP packet. The syntax is a hybrid of XML and NS-2 TR trace formats [10]. The *AGT* (agent), *RTR* (router) and *MAC* NS-2's literals relate to the application (END2END), network (NETWORK) and MAC (LAN) layers of the TCP/IP stack. The :r, :s, :f and :D literals have also been preserved to help understanding. They indicate whether the packet is received (:r), sent (:s), dropped (:D) or forwarded (:D) by the layer. However, contrary to NS-2, XAV does not log the payload and header fields in the operation records. Only the header IDs are written. These IDs reference the data stored in the data file. Thanks to these pointers, the packet information can be retrieved from anywhere in the operation file although it is only stored once in the data file. Figure 2 shows three pointers in the operation file referencing the same RTP object in the data file.



```
<?xml version='1.0'?>
<!DOCTYPE data SYSTEM "file:///tmp/data.dtd">
<data xmlns:mac='http://example.org/mac'>
   ...
  <rtp f='RTP258' >
       <pkt seqno='4'/>
  </rtp>
  <ip  f='IP258.1' >
       <pkt s='1' sport='0' d='21' dport='0' type='cbr' size='98' .../>
  </ip>
  <mac f='MAC258.1' >
       <Data s='1' d='5' duration='258' time='0.502897667'/>
  </mac>
   ...
</data>

<?xml version='1.0'?>
<!DOCTYPE operation SYSTEM "file:///tmp/operation.dtd">
<operation xmlns:mac='http://example.org/mac'>
...
  <nid n='1'>
     ...
     <AGT:s time='0.500000000' n='1' why='---'>
        <rtp f='RTP258' />
     </AGT:s>
     <RTR:r time='0.500000000' n='1' why='---'>
        <rtp f='RTP258' />
     </RTR:r>
     <RTR:s time='0.500000000' n='1' why='---'>
        <rtp f='RTP258' />
        <ip  f='IP258.1' />
     </RTR:s>
    <MAC:s time='0.502897667' n='1' why='---'>
        <rtp f='RTP258' />
        <ip  f='IP258.1' />
        <mac f='MAC258.1' />
    </MAC:s>
    ...
  </nid>
```

Figure 2. XAV data file (on top) & operation file (on the bottom).

Figure 3 extends the illustration of the pointer mechanism to the network scope. As we can see, the data are not only shared within nodes but along the whole packet path. As an example, the link/LAN layer header (in green in figure 3) is referenced by all MAC entities (transmitter and receivers) in the LAN. Data sharing is particularly useful for shared medium access networks and multicast protocols because data may be replicated hundreds times in these environments.

A XAV file can be transformed into an equivalent NS-2 flat trace file by recopying the header and payload fields of the data file into the corresponding operation tags. However, the reverse is not true: there is no secure method to convert an NS-2 TR file to XAV. Indeed, there are cases where the receive operations can not be surely reassociated to the corresponding send records, especially for packets that do not contain the sender address [1].

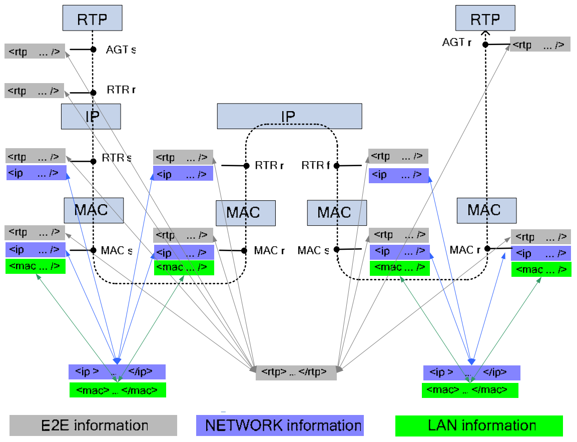


Figure 3. Sequence of tracing operations at the network scope.

Under XAV, all headers and data payloads whose content remains unchanged over the packet paths share the same ID. Several methods for distributing IDs can guarantee this property. The method that we propose is simple and well suited to mobile ad-hoc networks. An ID (or *headerID*) such as "MAC258.1" is the concatenation of a *protocolID*, a *bufferID* and a *nodeID*. *protocolID* differentiates headers from different protocols. In our current implementation, the set of *protocolID* values is {"RTP","AODV","IP","MAC"}. The protocol headers (and the payload) encapsulated in the same frame have the same *bufferID*. This value is returned by the *uid()* built-in function of the common packet header class of NS-2. The *'uid()'* static variable of NS-2 is incremented each time a new packet structure is allocated in the protocol stack. The uniqueness of the *uid()s* is reinforced particularly in the 802.11 MAC protocol for the RTS, CTS, ACK frames and for the AODV messages as the function which allocates frame buffers is not called for these packets. *nodeID* contains the MAC address of the node which sends the packet, when logging is performed at the RTR (NETWORK) and MAC (LAN) layers[2] and the null value when logging is done at the AGT (END2END) layer. *nodeID* is really useful only at the IP layer to recycle the *headerID* over the data end-to-end path. Indeed, the IP header can not be shared over the whole path

[2] The *nodeID* is set to -1 for the ACK and CTS packets.

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2008.4264
http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4264

because several of its fields are modified in the routers when forwarding the packet.

When a protocol receives a Service Data Unit (SDU) from an upper layer or sends a Protocol Data Unit (PDU) to a lower layer, XAV generates the *headerIDs* of all headers in the data unit and logs them along with the rest of the operation records in the operation file. An equivalent process is applied when a packet is received at a node. The headers content is logged in the data file at the transmitter side during the packet transmission in the protocol stack.

The implementation of XAV does not need to modify the code of NS-2 protocols[3]. Minor changes are introduced in the *ns-cmutrace.tcl*, *ns-mobilenode.tcl* and *ns-lib.tcl* scripts to interface the main TCL simulation script to the new C++ tracing module. In the tracing module, the major changes concern *basetrace.cc*, *trace.cc* and *cmu-trace.cc*. In *basetrace.cc,* we have added to the existing NAM and TR trace channels two additional Tcl channels to write in the operation and data files. Control functions have been added in *trace.cc* to open (attach), close (detach) and write into the XAV files from the main simulation script. The core of XAV is implemented in *cmu-trace.cc*. Actually, the interfacing of XAV is very simple because NS-2 tracing module is flexible and already supports other tracing frameworks such as NAM and TR.

3. DATA PATH NAVIGATION

Once the XAV files are put into the database, XQuery [8] is used to extract the data. The XQuery FLWOR expression allows to write join queries in a similar way to the familiar SQL select command. FLWOR stands for "for, let, where, order by, return", namely the five clauses that are used in the expressions :

- the *for* clause allows to set up an iteration over a set of XML nodes;
- the *let* clause allows users to declare intermediate variables;
- the *where* clause selects the nodes of interest;
- the *order by* clause sorts data;
- finally, the *return* clause tells how to compose and format the information to get back.

The *id()* built-in function allows to retrieve the header content from the XAV pointers in the operation file. On the same principle, the inverse function, *idref(),* is used to return all the send and receive operations related to a *headerID*.

Figure 4 shows examples of data paths that can be extracted from the *headerIDs* contained in a RTP packet. It is worth to note that multiple views can be elaborated from a single RTP path according the operating level of the post processing tools. For example, in figure 4, two paths (at least) can be associated to the RTP packet. The first one goes through the physical interfaces. It represents the *end-2-end path seen at the LAN level*. The second, namely the *end-2-end path seen at the E2E level*, only connects the RTP sender and receiver. A data path is described by a set of operations. A query returning the whole set of operations involved in a RTP path at the E2E level is presented in figure 5. XPath [11] is used to move along the node hierarchy of the XML trace. Its syntax is very similar to the one of Unix path names. The first three instructions declare the operating levels supported by the post-processing tool. The *id()* function[4] in the *for* clause finds all the RTP pointers in the operation file which reference the given RTP packet. The *let* clause retrieves the nodes operations from the RTP pointers.

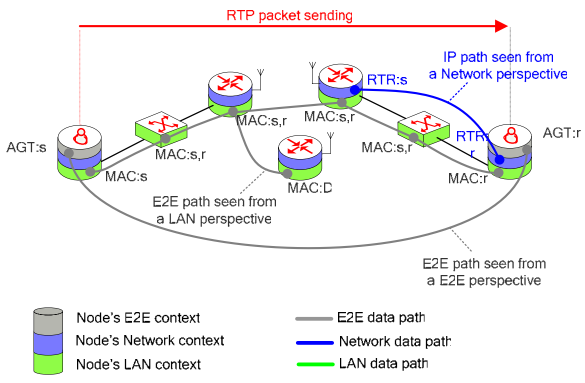


Figure 4. Examples of path views elaborated from a single packet path

The *where* clause puts in place the AGT (END2END) view. The :* symbol matches all types of node operations. It can be changed to :r or :s to select only the receive or send events. Finally, the *order by* clause sorts the operations in time order. The query returns the AGT:s and AGT:r operation records indicated in figure 4.

```
declare namespace MAC = "http://example.org/MAC";
declare namespace RTR = "http://example.org/RTR";
declare namespace AGT = "http://example.org/AGT";

for $pointerRTP in doc("operation.xml")/id("RTP0")
let $operationRTP := $pointerRTP/..
where $operationRTP/self::AGT:*
order by $operationRTP/@time
return $operationRTP
```

Figure 5. Example of query returning the operations involved in a RTP path at the E2E level.

The exploration service checks the nodes contexts to retrieve the packet paths. The context of the nodes that these paths traverse can be analyzed in their turn to exhibit relationships between flows. More details on node contexts are provided in the following sections.

4. INTERACTION WITH THE SIMULATION OUTPUTS

Under XAV, the trace files generated by simulators are not directly formatted as in figure 2. XAV raw outputs do not include *nid* tags. These tags are added afterwards during a process that we call "multidimensional formatting". Actually, this process is very fast because it parses the trace file in one pass and operates on a record per record basis.

[3] Excepted to guarantee the uniqueness of the distributed *uid()* values.

[4] Due to technical constraints, the current implementation under the MonetDB DBMS inverts the role of the *id* and *idref* functions.

The role of multidimensional formatting is to organize the trace file so that the locality of the queries can be exploited. Space, time and observation level are three fundamental elements in data exploration as they are commonly used abstractions for locating oneself in the environment. The space dimension is described in terms of physical entities such as nodes and networks. The time dimension is made of time intervals (minute, hour, day, etc). The protocol level dimension refers to the different layers of the protocol stack (LAN, Network, End to End). The interactive representation of the network is built in a way that the user can move and interact along these axes.

Multidimensional formatting restructures the simulation outputs in order to tie the interactive representation of the network to the content of the trace file. Space formatting is a simple process which involves grouping all operations related to a node in a node container and the node containers in network containers. Protocol-level formatting groups all operations related to a particular protocol level in protocol-level containers. Finally, time formatting arranges all operations occurring in the same periods in time containers. XAV implements the protocol-level, time and space containers as XML nested elements and scoped dimensions [12]. Compared to a file system tree, the advantage of a hierarchical XML structure is that all elements are enclosed in a common context and are easily accessible through XPath.

In the current implementation, the nodes operations are logged line-by-line in the operation file. The formatting program is a short script which parses the operation file in one pass. The operation records are filtered by node id (attribute *@n*), time (attribute *@time*) and protocol level (tag name). A temporary file hierarchy is generated during the XML trace scan as shown in figure 6. A directory is created for each node encountered in the operation file. A directory is created in the node directories for each protocol level used by the nodes. The third directory level gathers all operations occurring in the same time interval. At the end of the scan, the file hierarchy is transformed into an XML tree to form the final output. In the final output, an XML container is associated to each directory in the file hierarchy.

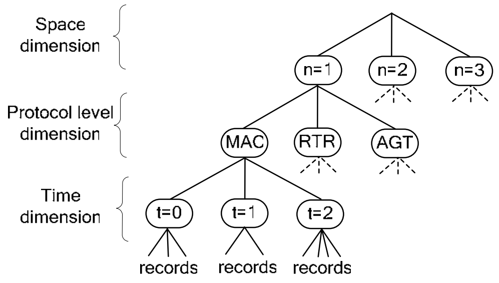


Figure 6. Example of file hierarchy generated during multidimensional formatting.

Figure 7 shows how the interactive representation of a subnetwork is bounded to XAV. The lower half of the figure represents the XAV output after the multidimensional formatting stage and the copy of the files into the XAV database. The blue spheres materialize the node containers. Each XML node container has it own XPath context which encloses all E2E, Network and LAN operations recorded by the network node during the simulation. Actually, these spheres correspond to the points from where the XQueries can attack the XAV database. The top of the figure represents the avatar the user interacts with.

As we can see the interaction points in the network representation have direct correspondences at the trace file level. These correspondences subsequently improve the time needed to retrieve the information of interest in the database.

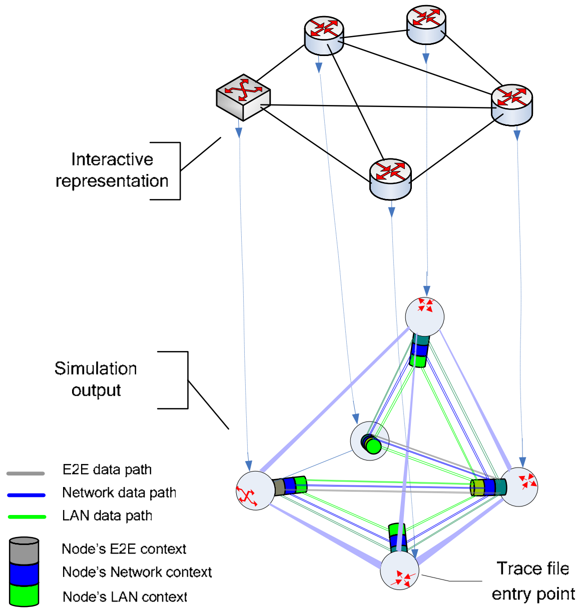


Figure 7. Multidimensional modeling of XAV outputs and interfacing with the network avatar.

Figure 8 illustrates the interaction between the database and the exploration service.

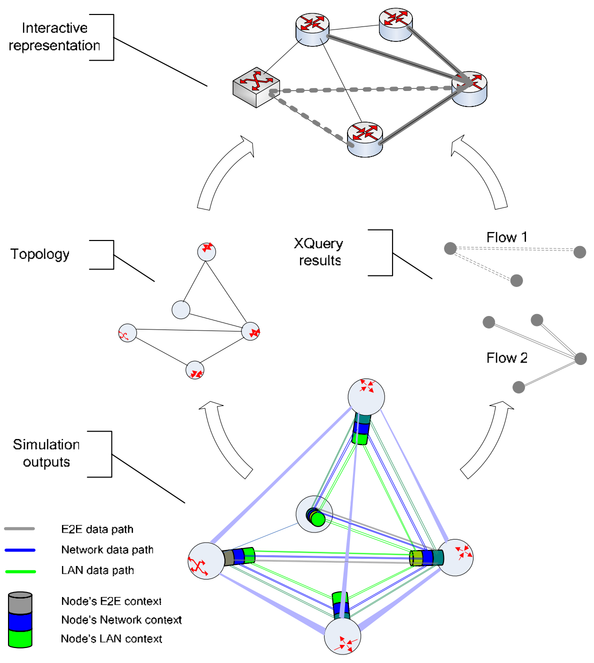


Figure 8. Interactions between the XAV database and the exploration service.

The explorer extracts the network structure from the simulation outputs along with the result of the XQuery. The interactive representation is built by combining the two pieces of information. The actual position of the network elements may correspond to the physical location of the objects or be random. Specific layout algorithms taking into account the traffic volume can also be used as in [6] and [7].

5. PERFORMANCE RESULTS

This section evaluates the performance of XAV for extracting a set of packet paths from the traces. The tests have been conducted on a mobile ad-hoc network consisting of 25 nodes placed on a regular grid. As illustrated in figure 9, the node transmission area includes all direct neighbors and the carrier sense area the two-hop neighbors. Ten CBR/RTP flows are established along the x and y axes of the grid. The CBR traffic starts at time 0. The CBR payload is 78 bytes long and each application generates a packet every 100ms. The AODV protocol is used to compute the routing tables. The routes used during the first second of simulation are depicted in figure 10. The 802.11 protocol operates at 2 Mbps.

We use a PC with 3GB of RAM with Linux and the MonetDB DBMS [13] (release “Feb 2008”). The MonetDB server which holds the XAV database runs on the local machine. The queries are sent through the Monet client interface. The resulting network representations (ie figures 11, 14 and 17) are drawn by hand. We use the trace format described in figure 2. Time and protocol-level formatting are disabled.

At the tracing level, contrary to the original logic, we log all the 802.11 packets decoded by the MAC even when the packets are not addressed to the MAC[5]. Logging these frames allows us to have a more complete model of the data paths and to reveal the intricate relationships between flows.

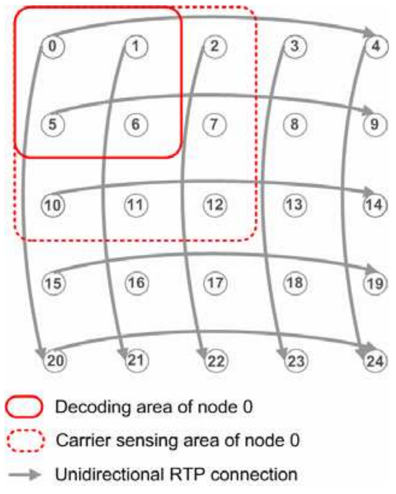


Figure 9. Traffic matrix and sensing areas.

[5] Only the packets originating from the decoding area are considered: as in the original framework, we do not log the packets from the carrier sensing area.

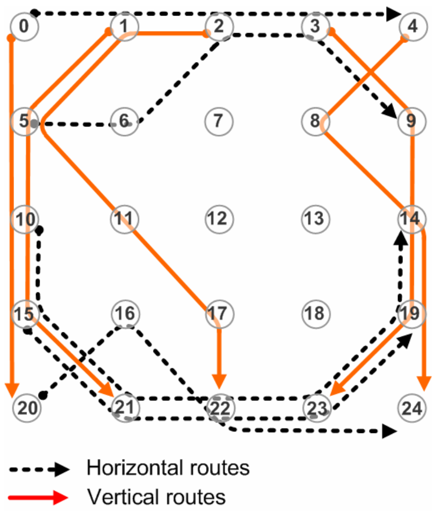


Figure 10. Routes maintained by AODV during the first second of the simulation.

The duration of the simulations varies from 1 to 100 seconds. One second of simulation generates about 8500 operations and 1 MB of trace. The operation file represents about 80% of the total output size. It is composed of 2.5% of AGT records, 12% of RTR records and 85.5% of MAC records on average.

5.1 Extraction of the paths originating from a CBR source

The first test simulates the request of a user who selects a CBR/RTP source in the interactive network representation and asks for the list of packet paths originating from this node. The horizontal paths returned by the exploration service are displayed in figure 11. The result shows that the routes are recomputed at time 24.7.

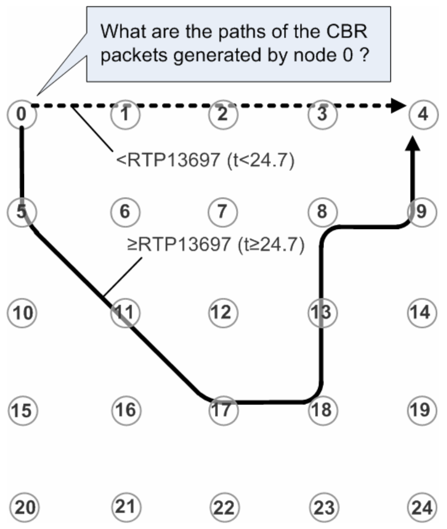


Figure 11. Horizontal paths followed by node 0 packets during the fifty first seconds of simulation.

The XQuery expression used in this example is detailed in figure 12. The query is the same than the one of figure 5 except the two nested *for* loops. The outer loop processes the end-to-end context to extract the whole set of RTP headers generated by node 0. The inner loop retrieves the packet paths from the header IDs.

```
declare namespace MAC = "http://example.org/MAC";
declare namespace RTR = "http://example.org/RTR";
declare namespace AGT = "http://example.org/AGT";

let $file := doc("operation.xml")
for $contextE2E in $file/operation/nid[@n=0)]/AGT:s/rtp/@f
return for $pointerRTP in $file/id($contextE2E)
let $operationRTP := $pointerRTP/..
where $operationRTP/(self::AGT:s|self::RTR:f|self::AGT:r)
order by $pointerRTP/@f, $operationRTP/@time
return $operationRTP
```

Figure 12. Query returning the paths followed by node 4 data packets at the network level.

The average response time of the nine CBR/RTP sources is graphed in figure 13[6]. The maximum values are obtained for node 0. The explanation is that node 0 runs two CBR applications (see figure 8) and thus generates twice as much packet paths. The minimum values correspond to the CBR sources that use the shortest paths. As we can see, the average response time for a 100MB file size is 1 second. This is the time needed to extract 1200 packet paths from the database. The query actually returns 6800 records, each path being described by 5.7 records on average.

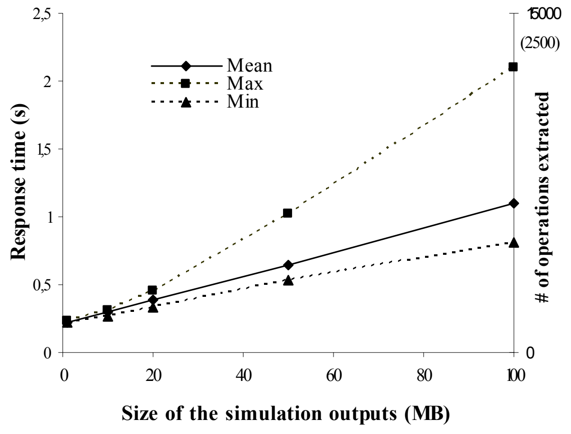


Figure 13. Average response time to extract the paths originating from a CBR source.

5.2 Extraction of the paths flowing through an IP router

The second test simulates the request of a user who selects a core node in the interactive network representation and asks for the list of flows forwarded by this node. An example of such request is represented in figure 14.

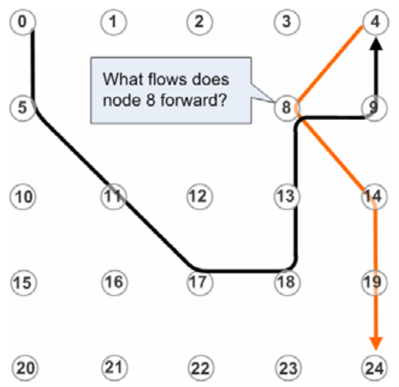


Figure 14. Paths flowing through node 8.

The query used in this example is presented in figure 15. The FLWOR is very similar to the query used to retrieve the flows originating from a node. The difference is that the outer loop has to process the IP context (instead of the end-2-end one) to retrieve the packets forwarded by the node.

```
declare namespace MAC = "http://example.org/MAC";
declare namespace RTR = "http://example.org/RTR";
declare namespace AGT = "http://example.org/AGT";

let $file := doc("operation.xml")
for $contextIP in $file /operation/nid[@n=8]/RTR:f/rtp/@f
return for $pointerRTP in $file/id($contextIP)
let $operationRTP := $pointerRTP/..
where $operationRTP/(self::AGT:s|self::RTR:f|self::AGT:r)
order by $pointerRTP/@f, $operationRTP/@time
return $operationRTP
```

Figure 15. Query returning the paths flowing through node 8 at the network level.

The average response time of the nine nodes in the center of the network is represented in figure 16. The maximum (respectively minimum) values correspond to the nodes with the highest (respectively lowest) traffic loads. Processing 100 MB takes 1.5 seconds on average. The result contains about 7400 operations which is the equivalent of 1200 packet paths.

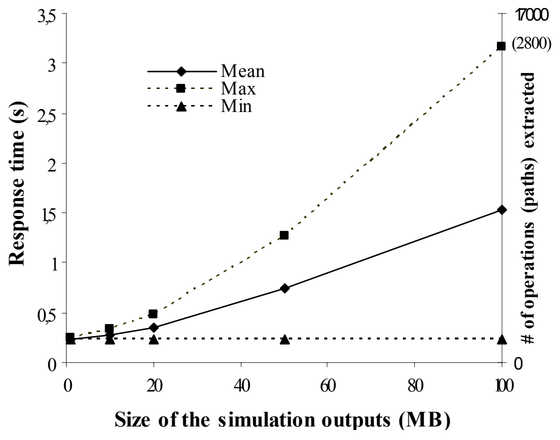


Figure 16. Average response time to extract the paths forwarded by the core nodes.

[6] The time needed to print the result is not considered. The time spent by MonetDB to shred the trace into main memory at first use is also ignored.

5.3 Extraction of the paths competing for the same medium

The last test simulates the request of a user who selects a node and asks for the list of flows competing for medium access at this node. Figure 17 shows an example of such request. It lists the set of flows competing for air access in node 8 vicinity.

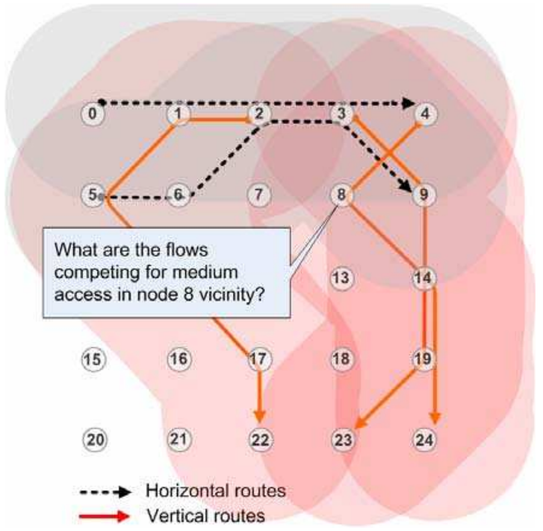

0
1
2
3
4
5
6
7
8
9
13
14
15
16
17
18
19
20
21
22
23
24
What are the flows competing for medium access in node 8 vicinity?
Horizontal routes
Vertical routes

Figure 17. Paths competing for medium access in node 8 vicinity.

Figure 18 presents the query corresponding to this request. As we can see, the query processes the MAC context to retrieve the data packets which have been received or dropped by node 8. The *distinct-value()* function filters the 802.11 frames retransmitted at the MAC layer. The condition on the *position()* function captures a fingerprint of the traffic at node 8 by keeping only one out of five MAC:D and MAC:r records.

```
declare namespace MAC = "http://example.org/MAC";
declare namespace RTR = "http://example.org/RTR";
declare namespace AGT = "http://example.org/AGT";

let $file := doc("operation.xml")
for $contextMAC in distinct-values($file/operation/nid[@n=8]/
    (MAC:D|MAC:r)[position() mod 5 = 0]/rtp/@f)
return for $pointerRTP in doc($file)/id($contextMAC)
let $operationRTP := $pointerRTP/..
where $operationRTP/(self::AGT:s|self::RTR:f|self::AGT:r)
order by $pointerRTP/@f, $operationRTP/@time
return $operationRTP
```

Figure 18. Query returning the paths competing for medium access in node 8 vicinity at the network level.

The average response time of the query of the 9 network core nodes is graphed in figure 19. The max (respectively min) values correspond to the most (respectively least) stimulated nodes at the MAC level. The average response time to process a 100 MB file is about 3.5 seconds. 18300 operations (which are equivalent to 3200 packet paths) are returned in this case.

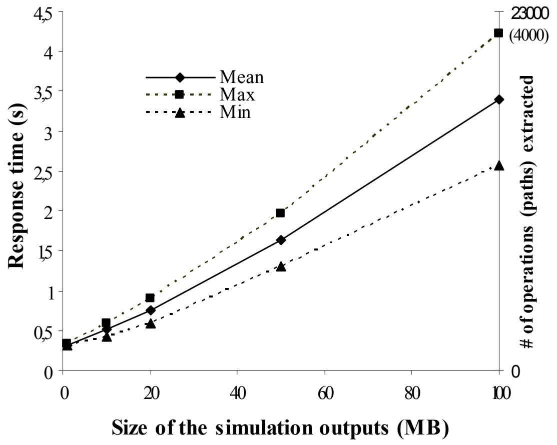


Figure 19. Average response time to extract the paths competing for medium access at the network level.

6. CONCLUSION

This paper has presented a general network tracing framework called XAV. XAV enables the exploration of large simulation outputs while offering users a more intuitive and enjoyable experience. For that purpose, XAV defines a multidimensional model to represent the trace data and quickly identify the data paths in the network. XAV has been implemented in the NS-2 simulator. Our solution offers high performance and flexibility levels. As future work, we plan to improve XAV IDs distribution algorithm and to adapt it to more complex encapsulation processes such as packet fragmentation. We will compare the performance with a SQL based solution using entity relation schemas instead of XML nested elements. A release of XAV will also be published in the framework of the Yavista project to serve as a basis for innovative development of real time applications in the network simulation area.

7. REFERENCES

[1] Scheuermann, B., Füßler, H., Transier, M., Busse, M., Mauve, M. and Effelsberg, W. 2005. Huginn: A 3D Visualizer for Wireless ns-2 Traces. In Proceedings of the 8th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (Montreal, Canada, October 10 - 13, 2005). MSWiM'05.

[2] JTrana, Available : http://ns2trana.googlepages.com

[3] Ben-El-Kezadri, R., Kamoun, F. and Pujolle G. 2008. XAV : A Fast and Flexible Tracing Framework for Network Simulation. In Proceedings of the 11th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (Vancouver, Canada, October 27 - 31, 2008). MSWiM'08.

[4] NS-2, Available: http://www.isi.edu/nsnam

[5] Cicconetti, C. Mingozzi, E. and Stea, G. 2006. An integrated framework for enabling effective data collection and statistical analysis with ns-2. In Proceedings of the 2006 Workshop on ns-2: the IP network simulator (Pisa, Italy, October 10, 2006). WNS2'06.

[6] Ben-El-Kezadri, R. and Kamoun, F. 2007. Towards MANET Simulators Massive Comparison and Validation. In Proceedings of the 18th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (Athens, Greece, September 3 - 7, 2007). PIMRC'07

[7] YAVISTA videos and tool,
Available: http://yavista.sourceforge.net

[8] XML, Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation (February 2004)
Available: http://www.w3.org/TR/XML

[9] XQuery 1.0: An XML Query Language, W3C Recommendation (January 2007)
Available: http://www.w3.org/TR/xquery

[10] Fall, K. and Varadhan, K. The ns Manual, The VINT Project

[11] XML Path Language (XPath) 2.0, W3C Recommendation (January 2007)
Available: http://www.w3.org/TR/xpath20/

[12] Bordawekar, R. and Lang, C. 2005. Analytical processing of xml documents: Opportunities and challenges. *SIGMOD Record*. 34(2), 27 - 32.

[13] Boncz, P. A., Grust, T. ,van Keulen, M.,Manegold, S., Rittinger, J., and Teubner, J. 2006. MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. In Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (Chicago, USA, June 27 - 29, 2006). SIGMOD'06.
MonetDB, Available: http://monetdb.cwi.nl