1-1-2011

An Open-Source and Declarative Approach Towards Teaching Large-Scale Networked Systems Programming

Taher Saeed  
*University of Pennsylvania*

Harjot Gill  
*University of Pennsylvania*

Qiong Fei  
*University of Pennsylvania*

Zhuoyao Zhang  
*University of Pennsylvania*

Boon Thau Loo  
*University of Pennsylvania*, boonloo@cis.upenn.edu

Follow this and additional works at: [http://repository.upenn.edu/cis_reports](http://repository.upenn.edu/cis_reports)

Recommended Citation  


This paper is posted at ScholarlyCommons. [http://repository.upenn.edu/cis_reports/946](http://repository.upenn.edu/cis_reports/946)

For more information, please contact repository@pobox.upenn.edu.
An Open-Source and Declarative Approach Towards Teaching Large-Scale Networked Systems Programming

Abstract
This paper describes our experiences at the University of Pennsylvania in developing course projects for a large advanced undergraduate and first year graduate course in networked systems. Students work in teams to develop substantial networked systems programming projects (>10000 lines of code) using network simulator 3 (ns-3), an emerging open-source network simulator that is aimed at replacing the popular ns-2 simulator. Projects are developed in layers, where students build upon earlier assignments, first developing a protocol for Internet Protocol (IP) routing, followed by a distributed hash table (DHT) overlay network, and finally, a keyword-based search engine. One novelty of our assignments is the use of ns-3 in a large class setting, where students navigating through hundreds of thousands of lines of existing code before adding their extensions. In addition, selected groups develop the final project using declarative networking, a novel declarative framework that allows protocols to be rapidly synthesized using a high-level logic language into ns-3 implementations.

Comments

This technical report is available at ScholarlyCommons: http://repository.upenn.edu/cis_reports/946
An Open-source and Declarative Approach Towards Teaching Large-scale Networked Systems Programming

Taher Saeed  Harjot Gill  Qiong Fei  Zhuoyao Zhang  Boon Thau Loo

University of Pennsylvania
(taher,gillh,qiongfei,zhuoyao,boonloo)@cis.upenn.edu

ABSTRACT
This paper describes our experiences at the University of Pennsylvania in developing course projects for a large advanced undergraduate and first-year graduate course in networked systems. Students work in teams to develop substantial networked systems programming projects (>10000 lines of code) using network simulator 3 (ns-3), an emerging open-source network simulator that is aimed at replacing the popular ns-2 simulator. Projects are developed in layers, where students build upon earlier assignments, first developing a protocol for Internet Protocol (IP) routing, followed by a distributed hash table (DHT) overlay network, and finally, a keyword-based search engine. One novelty of our assignments is the use of ns-3 in a large class setting, where students navigating through hundreds of thousands of lines of existing code before adding their extensions. In addition, selected groups develop the final project using declarative networking, a novel declarative framework that allows protocols to be rapidly synthesized using a high-level logic language into ns-3 implementations.

Categories and Subject Descriptors
C.2.1 [Computer-Communication Networks]: Network Architecture and Design, K.3.2 Computers and Education: Computer and Information Science Education

General Terms
Design, Experimentation, Language

Keywords
Networked Systems, open-source, implementation, experimentation, declarative networking

1. INTRODUCTION
The University of Pennsylvania has offered in the past four years, an implementation-oriented networked systems course titled Networked Systems [13]. This paper describes our experiences at teaching networked systems programming to a large class of 80+ students using ns-3, an emerging open-source network simulator. To our best knowledge, we are one of the first to attempt using the ns-3 platform for course projects that span network and application-layer protocols in a large class setting. Moreover, in the final project, we also experimented for a subset of students, the use of declarative networking [7], an innovative declarative approach towards building networking protocols.

1.1 Course Description
The Networked Systems course is cross-listed between our Computer and Information Sciences (CIS) and Telecommunications (TCOM) department, and is primarily geared towards advanced undergraduates or first-year graduate students. The course satisfies a project requirement for undergraduates, and is considered a technical elective for graduate students.

Throughout the semester, students cover a range of topics, ranging from networking fundamentals (Internet architecture, end-to-end systems design, IP routing, Transport layer, congestion control, quality of service, mobility), network programming techniques (event-driven vs thread-based programming, asynchronous network programming, performance measurement techniques), to emerging topics in application layer overlay networks. An overlay network is a virtual network of nodes and logical links that is built on top of an existing network with the purpose of implementing a network service that is not available in the existing network. Examples of overlay networks on today’s Internet include commercial content distribution networks such as Akamai, peer-to-peer (p2p) applications for file-sharing [5], content-based routing [1], and telephony [19], as well as a wide range of experimental prototypes running on the PlanetLab [16] global testbed.

The focus on the two latter topics differentiates this from typical networking courses, and comes at the expense of covering networking fundamentals at a faster pace, requiring students to read up topics in the textbook on their own (if they do not already have taken an undergraduate networking course).

Prior knowledge (at the level of undergraduate operating systems or networking course) is preferred but not required. However, given the scale of the programming assignments and the popularity of the class, students are advised to take the class only if they have had prior experiences in building large software systems. The class typically has an enrollment of around 80 students, making it one of the largest CIS/TCOM course at our university. In any given year, there are typically 50% CIS masters students, 30% TCOM masters students, and 20% undergraduates.

1.2 Motivation
In designing this course, one of the main challenges is in choosing an appropriate platform on which students can develop their course projects. Given that networking and distributed systems are mature topics, we were surprised to note that none of the existing
network simulator 2 [11] is another commonly used platform, but has become increasingly complex to use in recent years. This requires a steep learning curve for students if the projects are to span multiple layers of the network stack, hence is not feasible within the timeframe of a semester. Finally, one can use a router emulation platform [3], but this is more appropriate for projects that involve router configuration and is limited to projects at the IP layer.

1.3 Organization

The rest of the paper is organized as follows. In Section 2, we describe the ns-3 simulator, justifications of our choice, as well as describe some of our modifications to the code base to ease the software engineering process of students. Section 3 presents our course schedule, detailed project description, and experiences. Finally, we conclude in Section 4 with a discussion of our future directions in improving the course.

2. NETWORK SIMULATOR 3

This semester, we experimented with the use of the network simulator 3 (ns-3) [12]. The ns-3 project started in 2006, and is intended as an eventual replacement of the popular ns-2 simulator. It is currently under active development in the open-source community, and currently has over 500K lines of code. Prior to our class, there has been attempts [21] at using ns-3 in a class setting, but not at the scale (in terms of class size and scope of projects) that we have attempted.

2.1 Advantages of ns-3

In selecting an appropriate platform, we decided on ns-3, since it meets the following criteria:

**Realistic environment**: The ns-3 platform implements the existing network stack (interfaces/devices, link, network, transport, application). Hence, students can relate course materials (such as the Internet architecture, local area networks and inter-networking) to the actual software platform that they are building their projects on. The platform also exposes students to challenges of networked systems programming, which comes in the form of BSD-like sockets, IP addresses, multiple interfaces per-node, and handles real-world effects introduced by asynchrony, message losses and reordering.

**Well-organized and reusable code base**: One of our concerns in picking a realistic platform is the complexity of the code base that will result in a steep learning curve for students. Fortunately, ns-3 achieves a good balance between realism and complexity. Despite its realistic environment, the ns-3 code is well organized. In fact, the entire code base can be explained within a lecture (conducted by the TAs). ns-3 is based on a complete rewrite of the ns-2 simulator. It consists of an extensible framework, achieved via the use of modern C++ design patterns, such as component object model and factory design patterns. As a result, students are able to quickly understand the code and make modifications of their own, particularly in adding layers over existing protocols, replacing one protocol with another at one particular layer in the network stack.

**Combining simulations and actual implementations**: ns-3 has support for both simulation and emulation via the use of sockets. Students often found it easier to first simulate their protocols for a large number of nodes in a repeatable fashion to uncover protocol errors, and then deploy their implementations (with minimal changes except switching to an implementation mode) on an actual set of physical machines to study actual performance. This is not unlike the strategy adopted by systems and networking researchers/practitioners, and we aim to have our course projects closely mirror this process.

**Ease of configuration and testing**: We require that our platform runs on commodity PCs with no changes to underlying operating system, hence making it easy for system administrators to support these courses even when the class size is large, or when there are multiple project courses within the department. The default ns-3 emulation mode uses raw sockets, which requires sudo access to run on instructional machines (a problematic request with systems administrators). As a result, we rewrote the emulation platform using regular OS sockets, which allows a simple toggling between simulation and implementation without having to rewrite code. For testing purposes, ns-3 generates pcap traces, which allows one to use standard tools such as Wireshark [22] to analyze traffic traces for debugging purposes.

2.2 Course-Related Customizations

In addition to the addition of user-level sockets, we have made modifications to the ns-3 code base, to provide some initial code (approximately 5000 lines) that eases the process of students in designing and implementing their projects.

We provide an ns-3 application which serves as the main simulation program that executes scenarios to test students’ project implementations. This application takes as input, topology and scenario files, which allows students to set up networks configured using the Inet topology generator [6], control nodes joining and leaving the network, and allow selectively turning on of logging information in different simulated nodes/modules, and inject network traffic. Appendix A shows a basic scenario file which we released to students initially, and they were allowed to extend it (e.g. add new application modules to stress test their implementations).

As an alternative to running the entire simulation using the scenario file, we provide an interactive command prompt which allows students to interactively enter scenario commands while the simulation is in progress. Students found this feature extremely useful when debugging their protocols. Toggling between simulation and actual implementation is achieved simply by turning on a runtime flag that uses actual sockets (instead of ns-3 simulated sockets), and running our driver program on multiple physical machines.

3. COURSE DESCRIPTION

Table 1 shows a summary of our course schedule, listing the topics covered, as well as the schedule of our projects. In the first project, students develop IP routing protocols, and in the second project, students build an application layer peer-to-peer keyword search engine that uses the implementation from the first project. Students are given 3.5 weeks to complete each project, with an intermediate checkpoint in the middle of each project to ensure that
Table 1: Course Schedule

<table>
<thead>
<tr>
<th>Week</th>
<th>Material</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Internet Architecture</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Sockets, event-based programming, subversion</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Router, Switches, IP Routing</td>
<td>Project 1</td>
</tr>
<tr>
<td>5</td>
<td>IP Routing</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Transport (TCP and UDP)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Congestion control</td>
<td>Project 1 demo</td>
</tr>
<tr>
<td>8</td>
<td>Multicast protocols and overlay networks</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Distributed hash tables and DHT-based overlays</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Midterm</td>
<td>Project 2</td>
</tr>
<tr>
<td>11</td>
<td>Resilient and Mobility-based overlays</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Quality-of-Service</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Link-layer protocols</td>
<td>Project 2 demo</td>
</tr>
<tr>
<td>14</td>
<td>Wireless networking</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Special topics</td>
<td></td>
</tr>
</tbody>
</table>

Students stay on track. At the end of each project, students have to demonstrate their system primarily in ns-3 simulation mode. For extra credits, students can run the actual implementation on a number of physical machines in our departmental local cluster.

The first project was released to students on the 4th week of class, and due on the 7th week. The second project was released after the midterm on the 10th week, and due on the 13th week of class, two weeks before the final. To synchronize our course schedule with the projects, we opted to defer the coverage of traditional topics such as link-layer protocols (Ethernet, 802.11) and quality of service (QoS) routing until the later part of the semester. We also minimize the coverage of topics related to the physical layer (given that there are other classes at Penn that cover these topics in greater detail), and instead opted for spending more time on application-layer protocols. This decision turns out to be popular with students, who are able to relate course material to application-layer protocols (e.g. p2p search and Akamai), and also directly apply these concepts into their project implementations.

For both projects, students actively posted on our course news-group with questions that were addressed either by the teaching staff or other students. We asked students to refrain from posting on the ns-3 official mailing lists to avoid flooding the list with numerous code questions. In the rest of this section, we describe each of our projects in greater detail.

3.1 Project 1: Routing Protocol

In project 1, students implement two basic capabilities: neighbor discovery and routing. In neighbor discovery, each node has to determine its set of neighbors by periodically polling its network interfaces for the IP addresses of neighbors that are connected to it. As the topology changes, the set of neighbors of each node has to be updated in a timely fashion.

Given the neighborhood information at each node, the next part of the project involves implementing the link-state and distance-vector routing protocols. Link-state and distance-vector are typically used for computing routes within an ISP (or administrative domain), and are commonly known as OSPF (Open Shortest Path First) and RIP (Routing Information Protocol) in commercial implementations.

These protocols compute between any two nodes the path with least hop count. One of the challenging aspects of this project is the integration of the neighbor discovery and routing modules into the existing ns-3 framework. For instance, to do a neighbor discovery, each node would have to invoke a discovery message which involves a 1-hop UDP broadcast message on each of its interfaces. The routing protocol has to be compatible with the route advertisement framework of ns-3, and computed routes have to be used to instantiate forwarding tables used in ns-3.

In fact, our goal for students was to be able to replace any of the existing ns-3 routing protocols (e.g. OLSR [2] and AODV [15]) with their project implementations, and all transport protocols (e.g. TCP) and network applications that use these routing protocols have to continue working correctly as before.

As extra credits, students implement extra routing protocols (such as those used in mobile ad-hoc networks and delay tolerant networks), path-vector protocol (used as a basis for inter-ISP routing), performance metric-based routing, and an incremental Dijkstra algorithm that provides fast recomputation of routing entries at each node whenever the topology changes.

3.2 Project 2a: P2P DHT-based Search

In the next project, students implement a peer-to-peer search engine (PSearch) that runs over their implementation of the Chord DHT [20]. A DHT (distributed hash table) is an overlay network that provides a scalable lookup facility, i.e. given a key K, determine the node that stores the K can be achieved in $O(\log(n))$ hops while requiring each node to maintain only $O(\log(n))$ state, where n is the number of nodes in the network. Given that churn handling in DHT (i.e. repairing DHT routing tables in the presence of nodes joining or leaving the network, or node failure) is a challenging process, we simplified the project requirement by omitting node failures in the regular part of the project.

Students have to read the Chord paper on their own to implement the protocol. The Chord overlay is built as a layer over project 1’s routing protocol implementation. Students are allowed to choose either the distance vector or link state protocols that they developed in project 1 as the underlying routing protocol. If neither works, they are allowed to fallback on using one of ns-3’s default routing protocols (e.g. OLSR).

In addition to building the DHT, the project introduces students to the concept of layering in a network – the fact that a packet sent between two Chord nodes who are logically neighbors may traverse multiple IP hops is a concept that students appreciate much better once they have worked on the project.

PSearch is next developed as an additional layer over Chord as a p2p search engine, and used to build a DHT-based search engine [9] that provides a “google-style” search interface for retrieving all documents that matches a set of input keywords. Scalable lookups are performed based on inverted indices that are published and stored in the DHT, where the publishing key is the hash of each keyword (for the corresponding inverted index). Given a set of search terms $T_1, T_2, \ldots, T_n$, each search is executed as a distributed query that will use Chord as a routing substrate for retrieving the inverted indexes for all the search terms, and then performing a set intersection to retrieve the set of documents that contains all keywords.

As extra credits, students implemented more sophisticated ranking algorithms for searching, enhanced their implementations to support large inverted indices that cannot be fit into a single IP packet, churn handling in the presence of node failures, a Chord-based distributed file system, as well as implemented various over-
lay networks covered in class that provide support for resilient rout-
ing, mobility, and multicast at the application layer.

3.3 Project 2b: Declarative Networking

As the second option in project 2, we allow volunteer groups who did well in project 1, to explore the use of declarative net-
working techniques to implement the Content Addressable Net-
works (CAN) [18] DHT. Declarative networking is an exciting and
novel framework that has been proposed in the past few years in
the networking community. It uses a distributed database query lan-
guage for specifying protocols that can be implemented in only a
few lines of high-level logical statements. For example, traditional
routing protocols can be expressed in a few lines of code [10], and
the Chord DHT in 47 lines of code [8]. When compiled and ex-
cuted, these declarative networks perform efficiently relative to
imperative implementations, while achieving orders of magnitude
reduction in code size. Appendix B provides one example to illus-
trate the declarative networking language.

We explore this option on a pilot basis, to have a better under-
standing of the effectiveness of getting students to learn about a
protocol by implementing using high-level declarative abstractions.
Students utilized the RapidNet declarative networking engine [17],
which takes as input high-level declarative protocol specifications,
which are then compiled into ns-3 code. Our choice of CAN in-
stead of Chord for this option is due to the fact that there already
exists a declarative Chord implementation.

3.4 Evaluation

Student feedback from the use of ns-3 has been overwhelmingly
positive. In addition, we are encouraged by the performance of
students in our class. Students worked in groups of 3-4, and with
the exception of 2-3 groups, almost all groups have complete (or
near complete) implementations of both projects. In addition, sev-
eral groups completed at least one extra credit, with the best group
completing 10 extra credits (5 each for each project). On average,
groups wrote 7000 and 6000 lines of code for project 1 and 2 re-
spectively. Two groups volunteered for the declarative networking
option in the second project.

We briefly summarize some interesting observations we made in
the class:

Open source experience: One of our biggest concerns going into
the semester is the daunting 500K lines of code that students have
to navigate. To reduce this overhead, we pointed students to a few
directories that they should focus on. On hindsight, we found this
concern largely unfounded and very beneficial. Students were able
to use modern software development tools such as Eclipse to step
through large code bases. Code clarifications on our newsgroup
were quickly answered. The fact that an open source implementa-
tion has existing routing protocol implementations (e.g. OLSR
and AODV) proved tremendously helpful, as students can read ex-
sting implementations to understand the appropriate interfaces be-
fore adding their own implementations. This process also closely
mirrors the real world, where engineers often have to understand
and modify an existing complex code base.

Group dynamics: To ensure that all students contribute equally to
the project in a collaborative fashion, we made it mandatory that
students use the subversion repository right from the beginning of
project 1. In addition to allowing students to work concurrently
on different aspects of the project, subversion also provides a use-
ful mechanism via its logs that provides evidence of each student’s
participation. We further require that each individual filled in a
group evaluation form describing the contributions of each team-
mate. This form was filled in after the midterm and final where
no discussions were permitted, and used as a basis for adjusting
students’ final grades based on their contributions.

Declarative networking: One of the pleasant surprises this semester
was the success of the two volunteer groups in implementing the
CAN DHT using the RapidNet declarative networking engine. De-
spite some initial difficulties due to the leaning curve of Datalog,
both groups completed functional CAN implementations in 40+
lines of code, with one group completing additional enhancements
to the protocol outside the scope of the project. This is despite
the fact that these students do not have prior knowledge of declar-
ative networking, and most have not seen Datalog before. In fact,
students that used RapidNet were generally positive about their ex-
periences, and once they have mastered the language, would like to
use it for prototyping future protocols.

4. CONCLUSION

In this paper, we describe our experiences at introducing large-
scale programming assignments on networked systems using the
ns-3 platform. Our course has been a great success, receiving very
positive feedback from students. Within a span of 7 weeks, students
successfully completed the development of an IP routing protocol,
a complex DHT and a p2p keyword search engine on an open-
source platform. Moreover, two groups developed a DHT in 40+
rules using a declarative networking engine that we incorporated
into ns-3.

Moving forward, we plan to extend our initial work in the fol-
lowing directions. First, we plan to extend the projects for an ad-
vanced distributed systems course that we have been teaching for
the past few years. Here, students will develop using ns-3 a full-
fledged fault-tolerant distributed mail server (similar to gmail or
yahoo mail). This project has traditionally been developed using
Java, and we plan to explore (on a trial basis in the next offering)
the use of ns-3 platform, in particularly reusing the IP routing and
p2p keyword search capabilities already developed for this class.
Encouraged by our use of declarative networking technologies, as
a longer term agenda, we plan to introduce the use of declarative
networking into the networking curriculum from the ground up, as
a higher level abstraction for explaining protocol behavior, and de-
velop complete projects based on this framework.

Material and Source Code Detailed project descriptions (includ-
ing code documentation for our ns-3 extensions) are available for
viewing at [14]. Further materials, including initial release code for
students, solutions and grading scripts, are available to instructors
upon request.

5. ACKNOWLEDGMENTS

We would like to thank Penn graduate students Cheng Huang,
Changbin Liu, and Wenchao Zhou for giving valuable feedback on
our projects, and participating in the design and implementation of
the lab demonstrations. The development of the RapidNet declar-
ative networking engine is funded in part by NSF grants CAREER
CNS-084552, CCF-0820208, and IIS-0812270.

APPENDIX

A. EXAMPLE SCENARIO FILE

We show below a basic scenario file that we released to students
at the start of project 1. This scenario starts a number of nodes,
invokes the link state (LS) routing protocol, and then tests the pro-
tocol using an application-layer traffic generator. More details of
scenario configurations are available at [14].
Note that in the scenario configuration, we refer to nodes by node numbers, e.g 0,1,2. However, once the topology is initialized, our driver program assigns to each node multiple IP addresses. The reason why each node requires multiple IP addresses is that we wanted to emulate actual deployments where a router may participate (and inter-connect) different subnets. Our driver program then will select one of the IP addresses as the unique identifier of the node for the purpose of route computations in project 1.

```
# Turn on traffic traces for LS module on all nodes.
* LS VERBOSE TRAFFIC ON
# Turn on all traces for LS module on node 1
1 LS VERBOSE ALL ON
# Turn on traces for APP module on all nodes.
* APP VERBOSE TRAFFIC ON
# Turn on all traces for APP module on node 1
1 APP VERBOSE ALL ON
# Advance Time pointer by 15 seconds.
# Allow the routing protocol to stabilize.
TIME 15000
# Start traffic flow from node 1 to all the nodes.
1 APP TRAFFIC START *
# Advance time by 100 milliseconds.
TIME 100
# Stop Traffic flowing from node 1 to all the nodes
1 APP TRAFFIC STOP *
# Send PING from node 1 to node 8 (its neighbor).
1 LS PING 8 hello!
# Advance Time by 10 milliseconds.
TIME 10
# Bring down link between node 1 and 8
LINK DOWN 1 8
1 LS PING 8 hello2!
TIME 10
# Bring up link between node 1 and 8
LINK UP 1 8
1 LS PING 8 hello3!
TIME 10
# Bring down all links of node 1
NODELINKS DOWN 1
1 LS PING 8 hello4!
TIME 10
# Bring up all links of node 1
NODELINKS UP 1
1 LS PING 8 hello5!
TIME 10
# Send application level PING from node 1 to node 8.
1 APP PING 8 app_hello!
TIME 4000
# Dump Link State Routing Table.
1 LS DUMP ROUTES
# Dump Link State Neighbor Table.
1 LS DUMP NEIGHBORS
# Quit the simulator.
QUIT
```

## B. DECLARATIVE NETWORKING

The high level goal of declarative networks is to build extensible network architectures that achieve a good balance of flexibility, performance and safety. Declarative networks are specified using Network Datalog (NDlog), which is a distributed recursive query language used for querying network graphs. NDlog queries are executed using a distributed query processor to implement the network protocols, and continuously maintained as distributed views over one of the IP addresses and host state. Declarative queries such as NDlog are a natural and compact way to implement a variety of routing protocols and overlay networks. To illustrate, the following two NDlog rules compute all pairs of reachable nodes:

```
r1 reachable((S,D)) :- link((S,D)).
```

```r2 reachable((S,D)) :- link((S,Z)), reachable((Z,D)).
```

The rules r1 and r2 specify a distributed transitive closure computation, where rule r1 computes all pairs of nodes reachable within a single hop from all input links, and rule r2 expresses that “if there is a link from s to z, and z can reach d, then s can reach d.” By modifying this example, it has been shown in previous work [10] that we can construct more complex routing protocols, such as the distance vector and path vector routing protocols.

NDlog supports a location specifier in each predicate, expressed with @ symbol followed by an attribute. This attribute is used to denote the source location of each corresponding tuple. For example, all reachable and link tuples are stored based on the source address field. The output of interest (indicated by the rule query) is the set of all reachable((S,D)) tuples, representing reachable pairs of nodes from S to D. Note that the Query rule is not mandatory, in which case, all derived output tuples will be continuously computed and maintained in the network.

## C. REFERENCES


