A Simulation Environment for EIA-709.1/IP Routers

Stefan Soucek, Dietmar Loy
soucek@ict.tuwien.ac.at, dloy@coactive.com
Institute of Computer Technology, TU Vienna, Austria
Coactive Networks Inc., Sausalito, Calif., U.S.A.

***Abstract* – EIA-709.1/IP routers enable the extension of LonWorks networks over the Internet, using it as just another transmission channel. To investigate the influence of delay jitter and loss rates, this paper presents a simulation framework for both LonWorks nodes and routers. The Network Simulator ns-2 from UC Berkeley is extended by fieldbus nodes and EIA-709.1/IP routers. The simulation executes EIA-709.1 transport layer transactions using a sender/receiver model. Different router implementations can be analyzed using a generic scenario with sending nodes connected to local and remote receiving nodes.**

I. Introduction

LonWorks is one of several widely used fieldbus systems in the factory automation arena as well as in building automation. The system was originally developed by Echelon Corp., Palo Alto, Calif., in the early 90's [1], and its communications protocol LonTalk has been standardized in the U.S. by the EIA in EIA-709.1 [2]. The EIA-709.1 protocol features a layered architecture quite closely modeled after the OSI reference model and supports segmented networks connected by routers. The LonTalk transport layer provides a means for reliable end-to-end communication [3].

Both, its field of operation and the protocol properties of LonWorks, have driven the effort to inter-connect LonWorks networks over or with the Internet. More specifically, well established IP technology can be used to extend existing LonWorks segments acting as just another transmission media, or to access certain application-centric parts of the network remotely. The latter approach requires some kind of LonWorks/IP gateway, which acts on behalf of remote application-specific requests. A number of different concepts have been presented to access a control network from the application level [4]-[6]. In order to physically extend a LonWorks network, so-called *tunneling LT/IP routers* are used [7]. From a LonWorks perspective, they behave like ordinary LonWorks routers, from an IP perspective, they tunnel EIA-709.1 packets through an IP channel. The IP channel, however, behaves quite differently than native LonWorks channels [8]. In order to analyze the impact of packet loss, delay, delay jitter and re-ordering on such a channel, this paper presents a simulation framework that was created to host both LonWorks nodes and LT/IP routers.

II. The *ns-2* Simulation Framework

Among the desired features for a simulation framework are the ability to extend the framework using a commonly used programming language, the presence of existing modules that can model an IP network and its protocols, as well as some sort of scripting mechanism to create a number of different scenarios.

The *Network Simulator Version 2* (ns-2) [9] of the VINT project from UC Berkeley is being used for research including TCP flow control and multi-cast algorithms and, thus, provides substantial support to simulate IP related tasks. Furthermore, ns-2 is based on a C++ object hierarchy at its core, and is configurable by using an object-oriented Tcl variant, OTcl [10]. The C++ hierarchy is referred to as the *compiled* hierarchy, whereas the OTcl code is called the *interpreted* hierarchy. The interpreted hierarchy is mirroring C++ objects and also defines new aggregate objects, that combine objects from the compiled hierarchy. This architecture has two key advantages: (1) run-time demanding algorithms can be implemented using compiled code in C++, and (2) combination and configuration of the simulation objects can be done in the interpreted hierarchy without the penalty of a re-compilation for each simulated scenario or parameter set.

The main mechanisms to access compiled objects from OTcl are bound variables and a command handler method. A bound variable is accessible as an ordinary OTcl instance variable from the interpreted hierarchy. Anything, that is beyond the capabilities of simple variables, is implemented as an OTcl command which is passed to the compiled object's command method. The *connector* class makes use of these mechanisms to enable the aggregation of different ns-2 objects.

A typical ns-2 simulation consists of *node* objects, which are interconnected by *links* and exchange *packets*. *Agent* objects are attached to nodes and generate or consume traffic. The node object is responsible to receive packets from other nodes or one of its agents and send them over the corresponding link or to the addressed agent. The link itself models a certain queuing discipline, transmission delay and link errors (such as packet loss).

An extension of ns-2 to accommodate a new protocol basically needs to define new agent classes to handle that protocol as well as a set of simulator commands to conveniently set up scenarios using these agents. The following work is based on the methodology that was used to create an RSVP extension for ns-2 [11].

III. EIA-709.1/NS

The EIA-709.1 protocol allows nodes to communicate in a peer-to-peer fashion. Each node can send messages to any other node or group of nodes on the network. This means the communication relations on the network consist of a number of one-to-many (1:n) relations. Each of these relations can also be considered as n one-to-one relations. An appropriate model to describe such communication relations is a sender/receiver model. Therefore, EIA-709.1/ns implements two agent classes: a traffic source (Agent/Eia709) attached to a sending node and a traffic sink (Agent/Eia709Sink) attached to a receiving node.

The node objects needed to simulate fieldbus nodes have slightly different properties than typical ns-2 nodes. Fieldbus nodes do not implement routing functionality. Therefore, EIA-709.1/ns uses nodes of class *FieldbusNode*, which can attach to exactly one agent and transmit all sent packets to each of their neighbors. This way, the native broadcast nature of LonWorks channels can be modeled without having to use a special network abstraction. This approach neglects any bus arbitration times, but does consider transmission delays over the links. Figure 1 shows a simple setup with one sending node (subnet/node address 47/10) and three receiving nodes (47/11, 47/12, 47/13). In situations where media contention cannot be neglected, these nodes can also be connected to a network node, that simulates a multiple access protocol.

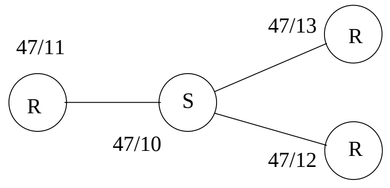

Figure 1 - Simple EIA-709.1/ns scenario.

The LonWorks agent attached to the fieldbus node implements the functionality of the EIA-709.1 transport layer. Since messages are broadcasted to all connected nodes, the agents have to perform address checking on the received messages and must drop those packets not destined for them. This is accomplished by supplying the subnet/node address information at creation time of the agent. By using the *set-nid* and *add-group* commands, agents can further be assigned a LonWorks node ID and added to groups, respectively.

The source agent provides an OTcl command to start a transmit transaction. This command takes parameters for a destination address, the transport service (unacknowledged repeated, acknowledged, request/response), the number of retries and the transmit timer t_{tx}. As in the Neuron-Chip, a priority and a non-priority transmit transaction can be issued in parallel. A transmit transaction in general involves a number of packets sent to the receiver (request messages) and replies from the receiver (confirmation messages). Depending on the service and addressing scheme, request messages can be normal data packets or reminders. Confirmation messages are usually acknowledgment packets from the receiver. The success or failure of a transmit transaction is logged to a trace file and registered in global transaction counters.

The sink agent processes the corresponding receive transactions, which are triggered by the reception of indication messages. Sink agents can be configured with an instance variable for the non-group receive timer t_{rx} (*nongroup-timer*), that decides for how long a receive transaction will be able to perform duplicate detection. Indication messages may be ordinary data packets or reminders. Depending on the used transport service the sink agent sends response messages back to the sender. The reception of a new receive transaction is logged to a trace file and registered in a global transaction counter.

On the sending node the packets are associated with a transmit transaction space $\mathcal{S}_\mathcal{T}$ and on the receiving nodes with receive transaction spaces $\mathcal{S}_\mathcal{R}$ as on the Neuron-Chip. Consequently, confirmation messages are matched by their priority attribute to a priority or a non-priority $\mathcal{S}_\mathcal{T}$. Considering the packet filtering based on the packet's destination address, a certain $\mathcal{S}_\mathcal{T}$ in the simulation is fully identified by a priority attribute and a source subnet/node address. Packets for the receiving nodes are matched by their priority attribute, source subnet/node address and destination address (not necessarily subnet/node) to a certain $\mathcal{S}_\mathcal{R}$. Each transaction in its transaction space ($\mathcal{S}_\mathcal{T}$ or $\mathcal{S}_\mathcal{R}$) is uniquely identified by a 4-bit transaction identifier (TID), if transmit timers, receive timers and packet delays are within certain limits. Unlike the Neuron-Chip, however, the simulation does not impose an upper limit on the number of active receive transactions per node.

If the timers are mis-configured or the packet delay suffers a high variance, TIDs may become ambiguous due to wrap-arounds and cause the receiving node to either let a duplicate packet start a new transaction or treat a packet of a new transaction as a duplicate of an old one. Both situations result in protocol failures that the simulation catches by tracking a simulation-wide unique transaction identifier (Check-TID). Such error situations are logged to a trace file and registered in global transaction counters. Table I lists the available transaction counters, which provide statistics across all nodes in the simulation.

Table I
TRANSACTION COUNTERS.

cnt-trans-ok	completed transmit transactions
cnt-trans-fail	failed transmit transactions
cnt-trans-rx	indicated receive transactions
cnt-failures-tx	failures in transmit transactions
cnt-failures-rx	failures in receive transactions

The global transaction counters are implemented as OTcl class variables of the source and sink agents and can be accessed by their corresponding class procedures. The class variable concept has the advantage to be independent from the actual number of nodes in the simulation.

IV. LT/IP Router Framework

In order to investigate the influence of an IP channel between two LT/IP routers, the LonWorks nodes (using the broadcast paradigm) have to be connected to the IP network (using routing) over the LT/IP router. A typical scenario contains a sending node that communicates to local receivers as well as remote receivers. A remote receiver is considered a receiver behind the IP channel. A simple setup is depicted in Figure 2, consisting of two local and one remote receivers, denoted by superscripts l and r, respectively. The routers are referred to as sending router, when it is connected to the sending node's network, or receiving router, when connected to the remote receiving node's network.

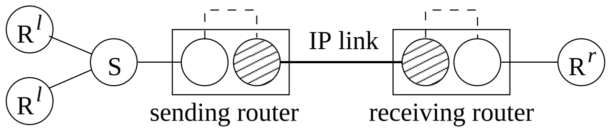

Figure 2 - Simple LT/IP router scenario.

The LT/IP devices comprise a fieldbus node to communicate to LonWorks nodes and an ordinary ns-2 node (drawn hatched in Figure 2) to connect to an IP link. To the fieldbus node, a simple agent is attached that forwards all packets to an LT/IP agent, which is attached to the IP node. The LT/IP agents come in two flavors, depending on whether they are connected to the sending router (Agent/LTIP/Send) or the receiving router (Agent/LTIP/Recv). This method has the advantage to re-use the existing node classes in the router and, thus, get the full support of ns-2 IP simulation. The core LT/IP functionality is encapsulated in the LT/IP agents, which act as agents for the LT/IP protocol defined in [7]. Finally, the sending and receiving LT/IP agents have to be logically connected to establish ns-2 IP communication between them.

The routers are connected by an ns-2 duplex-link object, which can be configured with a certain transmission delay, can include an error module for simulating packet losses, and a special delay module for simulating delay jitter. The loss error module can be configured for either a certain bit-error rate or a packet loss rate. The delay module has a random variable attached, which varies the extra delay for each packet transmitted over the link. The random variable can be chosen from a number of probability distributions, including constant value, uniform distribution, and exponential distribution.

Using a constant value, the simulation can on purpose produce certain delay jitter situations, which cause a protocol failure in the corresponding transaction spaces, if the LT/IP routers include no provisions to detect stale packets. The application pattern on the sending node can also be constructed such to produce TID wrap-arounds, and make the sending node use the very same TID for a certain destination as for the last transaction to this node. These two cases, the stale packet and the TID wrap-around, are the two critical effects that have to be overcome by LT/IP routers.

To further investigate the effects of different LT/IP router implementations, the LT/IP agents are subclassed for the different approaches. There exists a plain LT/IP agent that just forwards packets, a subclass that sequences the packets and drops out-of-order packets and a subclass that drops packets due to their time-stamp. Other algorithms to detect stale LT/IP packets can be researched by creating a new subclass and attaching the resulting agents to the LT/IP router framework.

The effectiveness of each approach is summarized by statistic counters in the routers, one for dropped by sequencing, number of stale requests, number of stale confirmations, total number of requests, and total number of confirmations. Together with the statistic counters for the nodes, it can be seen, how many error situations occurred and how many of them were detected or led to protocol failures in the nodes.

V. Current Work using EIA-709.1/ns

Using the basic simulation framework for LonWorks nodes and LT/IP routers, a more detailed investigation on the following items can be made,

- New algorithms to detect stale LT/IP packets in LT/IP routers,
- a large-scale traffic simulation with a configurable number of nodes,
- and a model for LonWorks bus arbitration (predictive p-persistent CSMA).

The basic LT/IP agents use a straight forward way to detect stale LT/IP packets: they drop out-of-sequence packets and packets whose time-stamp is too far in the past. The time-stamping method is easy to implement in the simulation, because simulation time is always available. In the Internet, no global time base is generally available. Nodes have to use protocols such as NTP [12] to synchronize their clocks. This can only be done to a certain accuracy. Current work is focused on the development of other algorithms to detect stale LT/IP packets, without the need to use accurate clock synchronization and time servers. With the EIA-709.1/ns ability to induce packet losses and delay jitter, the different algorithms will be compared performance-wise and resource-wise.

Any of the available LT/IP agents can be tested in a simple scenario, which is constructed to produce very well known error situations. The performance, memory usage and test coverage, however, can only be approached using a large simulation setup with an arbitrary number of nodes. A generic model for such a setup is shown in Figure 3.

This model contains n communication relations $\mathcal{R}_i$, each consisting of one sending node S_i, j_i local receivers R^l_{i,λ_i} ($\lambda_i = 1 \ldots j_i$) and k_i remote receivers R^r_{i,ρ_i} ($\rho_i = 1 \ldots k_i$). Furthermore, the sending nodes follow application patterns $\mathcal{A}_i$ to generate the traffic. The key task here is to find a set of parameters n, j_i, k_i ($i = 1 \ldots n$) along with some typical critical application patterns $\mathcal{A}_i$ to verify the functionality of the LT/IP routers. Critical application patterns in this respect are applications that will cause misleading TID wrap-arounds for certain destinations. A typical example for this is the periodic polling of values from 15 destination nodes

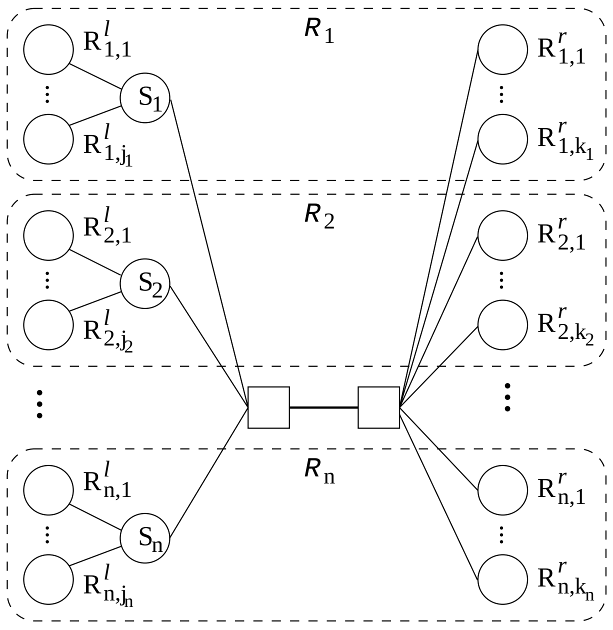

Figure 3 - Generic LT/IP router scenario.

in a round-robin fashion.

Finally, simulation of the local LonWorks segments can be improved by using the ns-2 network node. This is a special node that simulates a shared medium access and uses configurable modules for media contention resolution and link layer queuing. By implementing the predictive p-persistent CSMA algorithm for the contention resolution module, the shared media access on typical LonWorks channels can be simulated. This is useful to approximate the maximum transaction rate through LT/IP routers on a realistic channel, which is limited by the bit-rate, packet size and the channel arbitration time.

VI. Conclusion

A simulation framework for fieldbus and more specifically EIA-709.1 networks to IP network connectivity was presented. Such a framework is fundamental for developing fieldbus to IP routers optimized in functionality, performance, and cost. In parallel to this effort a PowerPC based hardware platform is under development in order to verify the simulation results under real-time and real-world conditions. Networking in industrial and building automation as well as residential areas is highly cost sensitive and the hardware resources available need to line up almost perfectly with the minimum application requirements in order to reach the market expectation. Even higher integration such as System-On-Chip (SOC) designs will be required for further cost reduction and to satisfy the complexity of fieldbus to IP connectivity.

Further research areas will include IP connectivity for other fieldbus protocols such as CEBus (EIA600), EIB, P-NET, and Profibus, as well as defining criteria for the ideal fieldbus protocol that directly supports IP connectivity over WANs. Industry and universities are working close together to create standards and to improve existing solutions.

References

[1] Motorola, *LonWorks Technology Device Data*, 1997, Rev 4.

[2] Electronic Industry Alliance, *Control Network Specification*, March 1998, EIA STANDARD EIA-709.1.

[3] D. Dietrich, D. Loy, and H.-J. Schweinzer, *LON-Technologie: Verteilte Systeme in der Anwendung*, Hüthig Verlag, Heidelberg, 1997.

[4] Martin Knitzak, Martin Kunes, Martin Manninger, and Thilo Sauter, “Applying Internet Management Standards to Fieldbus Systems”, in *Proc. of the IEEE International Workshop on Factory Communication Systems*, Piscataway, 1997, pp. 309–315.

[5] OPC Foundation, *OPC Data Access Custom Interface Specification 2.0*, July 1999, http://www.opcfoundation.org/ opc_spec_document.htm.

[6] Stefan Soucek and Hans-Jörg Schweinzer, “Considerations on a LonWorks/IP Gateway Implementation”, in *Proc. of FeT’99*, Magdeburg, Germany, September 1999, pp. 291–298.

[7] Ed Koch, *Lontalk IP Channels*, LonMark Routers Task Group, 1999, Request for Comments. Version 1.

[8] Stefan Soucek and Dietmar Loy, “Control Network Data over IP: An Architectural Overview”, *Fachzeitschrift it+ti Informationstechnik und Technische Informatik*, vol. 42, no. 4, 2000, Oldenbourg Verlag München.

[9] K. Fall and K. Varadhan, *ns Notes and Documentation*, April 1998, http://www-mash.cs.berkeley.edu/ns/ nsDoc.ps.gz.

[10] David Wetherall and Christopher C. Lindblad, “Extending Tcl for Dynamic Object-Oriented Programming”, in *Proc. of the Tcl/Tk Workshop 95*, Toronto, Ontario, July 1995.

[11] Marc Greis, *RSVP/ns: An Implementation of RSVP for the Network Simulator ns-2*, University of Bonn, http://www-student.informatik.uni-bonn.de/∼greis/rsvpns/rsvpns.ps.gz.

[12] D. Mills, *Network Time Protocol (Version 3) specification, implementation and analysis*, University of Delaware, March 1992, RFC-1305.