STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

COLLECTIVE COMMUNICATION - A SOLUTION FOR RELIABLE NETWORK MANAGEMENT

Marius IURIAN[**]

Received: September, 4, 1995
AMS subject classification: 68M10, 68Q10.

Rezumat. - Comunicaţiile de grup - o soluţie pentru o gestiune a reţelelor cu grad ridicat de siguranţă. Lucrarea de faţă prezintă o soluţie pentru construirea unor aplicaţii de monitorizare şi gestiune a reţelelor de calculatoare care să fie tolerante la erori, cu un grad ridicat de siguranţă. Pentru aceasta se propune utilizarea comunicaţiilor de grup, într-o variantă derivată din modelul folosit de sistemul ISIS.

1. Introduction. The Simple Network Management Protocol (SNMP) helps network managers locate and correct problems in a TCP/IP network. Managers run a SNMP client on their local workstation and use the client to contact one or more SNMP servers that are running on remote machines. Implicit in the SNMP architectural model is a collection of network management stations and network elements. Network management stations execute management applications which monitor and control network elements. Network elements are devices such as hosts, gateways, terminal servers, and the like, which have management agents responsible for performing the network management functions requested by the network management stations. The Simple Network Management Protocol (SNMP) is used to communicate management information between the network management stations and the agents in the network elements.

All implementations of the SNMP must support the next five operations:

GetRequestPDU	Fetch a value from a specific variable
GetNextRequestPDU	Fetch a value without knowing its exact name
GetResponsePDU	Reply to a fetch operation
SetRequestPDU	Store a value in a specific variable

[**]*"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

TrapPDU Reply triggered by an event.

The first four operations are used to obtain or to set the value of the variables maintained by the SNMP servers and, in general, are less critical in time. For the Trap operation the time is very important because this operation is triggered by the occurrence of some specific event (for example a network card stopped functioning at time T). In fact the Protocol Data Unit (PDU) for the Trap operation contains a field that indicate the time of the event's occurrence.

The implementations of SNMP - servers or clients - are based on UDP (User Datagram Protocol) as a transport protocol. UDP provides connectionless communication among application programs. This make the protocol simpler and therefore with a greater performance as speed in comparison with a connection oriented protocol (like TCP - Transmission Control Protocol). The greatest disadvantage of UDP is that it is unreliable: it is possible to loose datagrams or these can arrive out of order or just after a great amount of time.

For these reasons we consider that the first four operations of SNMP can be implemented very well using UDP but an UDP based implementation of the Trap operation can be very inefficient and unreliable.

Other problems that should be mentioned here are the crash of the network management application or of the site which is running this application or the failure of the underlying network. These problems affect the availability of the network management service.

Another aspect that we want to mention before trying to evaluate different solutions is that the configurations of the sites that are being monitored and that are emitting SNMP Traps are sometimes difficult to modify. For example, a Novell Netware server running a SNMP server have only the possibility to send traps to a specified IP address, it cannot send this trap to a group of sites and cannot be modified very easy to do that (that modification implies the rewriting of the entire SNMP.NLM provided with the system, and this is not a trivial task).

2. Different solutions. The User Datagram Protocol (UDP) is unreliable if we consider an internetwork environment, but works very well in a local network environment (that is the principal reason for choosing UDP as transport protocol for the implementation of NFS - Network File System). This thing suggests the use of a **proxy** - a site which is in the same local area network as the monitored server. So, instead of instructing the server to send traps to a network management station, possibly situated outside the server's LAN, the server should send all the traps to this special site named proxy. The name was chosen because there is a great similarity with the proxies used by SNMP for the network objects that doesn't support directly SNMP (see [Stallings1993]).

Now that the traps sent by the servers arrived at the proxy, it is its responsibility to deliver the traps to the management station. A first approach can be to use a reliable transport mechanism (like TCP) to deliver the traps but this solution is still unable to overcome errors like site crashes and network failures. To achieve **availability** the network management station must be **replicated.** It is still possible to have the proxy responsible for sending copies of the trap to each of the network management station replica but it is important to remember that the ordering in time of such traps is very important. Also, the contents of the group of replicated management stations can vary in time (because of site crashes or network partitioning).

These are the reasons for choosing a model for **collective communications** in the implementations of the proxies and the management stations.

3. The proposed Model. The system is composed by a set of processes $P = \{ p_1, p_2, ..., p_n \}$ each one having a disjoint memory space. The processes represent the proxies and the network management stations. It is presumed that this set contains all the processes needed and it is known in advance. The processes failures follows the model **fail-stop**, which means that after the apparition of a failure the process stops all its activity (that means, more precisely, that in case of a failure a process stops immediately sending or receiving messages). The network can be partitioned due to link failures, messages can be lost, delayed, duplicated

or delivered out of order. The processes are structured in a set of process groups $G = \{ g_1, g_2, ..., g_m \}$. Each process group has a name, a set of component processes and a unique special process x which is named proxy (so each group is composed by a proxy, a number of network management stations and all their replicas) :

$\forall g_i \in G, g_i = \{ p_{i1}, p_{i2}, ..., p_{ik} \} \subseteq P.$

$\forall g_i \in G, \exists! p \in g_i,$ p will be denoted proxy(g_i)

$X = \{ x \in P \mid \exists g \in G$ s.t x=proxy(g)$\} \subseteq P$ the set of proxies .

Any process can join or leave a group in any moment. A process take notice of a change in the contents of a group (to which it belongs) by using the notion of **view.** A view of a group is the list of its members. A **view-sequence** for a group g is an array: $view_0(g)$, $view_1$ (g), ..., $view_n$ (g) with the following properties:

$view_0$ (g) = 0,

$\forall i$: $view_i$ (g) $\subseteq$ P,

$view_i$ (g) and $view_{i+1}$ (g) differs by exact a process.

A process take notice of a failure of some processes in the same group just by using this view-sequence. This model suppose that there always exist the possibility of direct communication between two processes. The transport level can offer two types of communications: multicast messages and point-to-point messages.

It is defined further the relation in time between different events (like sending and receiving messages) using the model proposed by L. Lamport in its very important paper "Time, clocks, and the ordering of events in a distributed system" (see [Lamport1978]).

DEFINITION 1: The process execution is a partially ordered sequence of events, each event corresponding to an atomic action. By " $\rightarrow_p$ " it will be denoted the acyclic order between two events that occur in process p.

The following notations will be used further in this paper:

$send_p$ (m)	the event of sending the message m by the process p to one or more processes globally designed by dests(m).

$rcv_p(m)$	the event of receiving the message m by process p.
$rcv_p(view_i(g))$	the event by which process p take notice about the group g contents (group g including process p).
$deliver_p(m)$	the event of delivering the message m received before by process p.

If a process is a member of multiple groups then it must be indicated also the group to which a message is sent, received or delivered. The notation $deliver_p(m,g)$ means the delivering of message m to process p as a member of group g.

DEFINITION 2: The transitive closure of the relation "$\rightarrow_p$" will be denoted "$\rightarrow$" and it will be an ordering relation having the following properties:

If $\exists\ p \in P$ so that e "$\rightarrow_p$" e' then $e \rightarrow e'$.

$\forall m$: $send_p(m) \rightarrow rcv_q(m)$.

Two distinct events a and b are **concurrent** if and only if we don't have neither $a \rightarrow b$ nor $b \rightarrow a$. For the messages m and m' the notation $m \rightarrow m'$ will have the significance of send(m)$\rightarrow$ send(m').

Many models represent the relation $\rightarrow$ using timestamp vectors.

DEFINITION 3: Let $VT(p_i)$ be the **timestamp vector** for a process p_i, an array of length n (where $n = |P|$) indexed by the process identifier. The rules for computing the timestamp vector are the following four:

$VT(p_i)$ is initialized with 0 when the process p_i starts.

For every event send(m) from p_k to p_i the component $VT(p_k)[i]$ is incremented by 1.

Every message sent by process p_i in multicast mode will contain the updated timestamp vector.

When a process p_k deliver a message m received from process p_i which contains the timestamp vector VT(m), will modify its own timestamp vector following the next rule:

$\forall j \in \{1, 2, ..., n\}$: $VT(p_k)[j] := \max (VT(p_k)[j], VT(m)[j])$.

So, the timestamp vector contained in a message counts the number of messages, calculated relative to each sending process, that causally precedes the message m. The

timestamp vectors are compared using the following rules:

$VT_1 \leq VT_2$ if and only if $\forall i: VT_1[i] \leq VT_2[i]$

$VT_1 < VT_2$ if $VT_1 \leq VT_2$ and $\exists i: VT_1[i] < VT_2[i]$

It can be easily proven that $m \rightarrow m'$ if and only if $VT(m) < VT(m')$.

The systems that implement group communication usually support three types of events ordering:

Causal ordering - which is the order defined before and represented using timestamp vectors.

Forced ordering - which means that a given sequence of events is occurring in the same order at every member of the group.

Immediate ordering - which means that every event is occurring in the same order at every member of the group, relative to every other event in the system.

It is obvious that the second order is stronger than the first and the third order is stronger than the second (and, of course, the second and the third orders are more difficult to implement, requiring more message exchanges between the processes).

ISIS system [Birman1991] supports all the three operations through three types of protocols: CBCAST, ABCAST and GBCAST and also the model proposed by Ladin-Liskov [Ladin1992] supports causal, forced and immediate operations. For our purpose it is sufficient the causal ordering and a protocol very similar to CBCAST protocol. In the fourth section we will indicate how this solution can be implemented using ISIS system version 2.1.

The protocol implements the following causal order:

(1) $\forall m, m', x \in X: send_x(m) \rightarrow send_x(m') \Rightarrow \forall p \in dests(m) \cap dests(m'):$

$deliver(m) \rightarrow_p deliver(m')$.

The protocol for implementing causal ordering (given in formula (1)) in our case is described by the following rules:

Before sending the message m, the process $x_i \in X$ is incrementing $VT(x_i)[i]$ and insert the updated timestamp vector in message m.

The process $p_k \neq x_i$ ($p_k \notin X$) which receives the message m sent by the process x_i containing the timestamp vector VT(m), delays the deliver of m until the following condition become true: $\forall j \in \{1, 2, ..., n\}$ $VT(m)[j] =$

$VT(p_k)[j] + 1$ if $j = i$ and $VT(m)[j] \leq VT(p_k)[j]$ otherwise.

When a message m is delivered the timestamp vector $VT(p_k)$ is updated according to the rules mentioned in definition 3.

Theorem: The protocol described before is correct - that means it respect the two properties of **safety** (it respect the causal order) and **liveness** (it will not infinitely postponing the deliver of any message).

Proof: Because the processes sending messages are members of X (the set of proxies) and the processes receiving messages are from $P \setminus X$ it result that messages sent by different proxies are not causally related. That means that for two distinct messages m_1 and m_2 sent by a process x_i (in this order) and arrived at process p_k will have the timestamp vectors in the following relation: $VT(m_1) < VT(m_2)$. Applying the second rule of the protocol we will have that message m_2 will be delivered only after the deliverance of the message m_1 and so the safety is proved.

For the proof of liveness let suppose that it exist a message m sent by the process x_i and that it cannot be delivered to process p_k. From the rule 2 of the protocol we will have one of the following relations true:

(2) $VT(m)[i] \neq VT(p_k)[i] + 1$.

(3) $\exists j \in \{1, 2, ..., n\}$ $VT(m)[j] > VT(p_k)[j]$ where $j \neq i$.

The relation (2) means that m is not the next message to be delivered from x_i to p_k. Because the number of messages preceding m is finite results that exist a message m' sent by x_i, received by p_k and not yet delivered which the next message to be delivered (so m' satisfies the negation of (2)). If m' is also delayed we will obtain a contradiction.

Let consider now the relation (3) as being true. Let $n = VT(m)[j]$. The n-th transmission from process x_i must be a message m' which satisfy $m' \rightarrow m$ and which was either not received by p_k or it has been received and delayed. We can now repeat this reasoning for m' and because the number of messages transmitted before m' is finite and the relation $\rightarrow$ is acyclic it will result a contradiction.

4. Final remarks. The model proposed solve the raised problem in all its aspects offering a reliable solution for network management using SNMP Traps.

First of all it does not require any modifications on the monitored servers due to the use of proxies. The groups includes only the proxies and the network management stations

The replication of the network management stations and the use of a special protocol for collective communication make the solution reliable and performant.

For implementing the proxies and the network management applications it is possible to use ISIS the toolkit for distributed programming developed at Cornell University by a team lead by K. Birman [Birman1990]. The advantages of ISIS are the different types of groups organizations (peer-to-peer, client-server, diffusion and hierarchical) and the variety of protocols for group communication [Birman1993]. The presented model can be implemented using the diffusion type of group organization and the CBCAST protocol.

REFERENCES

[Birman1990] K. BIRMAN, R. COOPER, T. JOSEPH, K. MARZULLO *"The ISIS System Manual"*, 1990.

[Birman1991] K. BIRMAN, A. SCHIPER, P. STEPHENSON *"Lightweight Causal and Atomic Group Multicast"*, ACM TOCS, Vol. 9, No. 3, 1991, pp. 272-315.

[Birman1993] K. BIRMAN *"The Process Group Approach to Reliable Distributed Computing"*, CACM, Vol. 36, No. 12, 1993, pp 36-54.

[Ladin1992] R. LADIN, B. LISKOV *"Providing High Availability Using Lazy Replication"*, ACM TOCS, Vol. 10, No. 4, 1992, pp. 360-392.

[Lamport1978] L. LAMPORT *"Time, Clocks, and the Ordering of Events in a Distributed Systems"*, CACM, Vol. 21, No. 7, 1978, pp. 558-565.

[Stallings1993] W. STALLINGS *"SNMP, SNMPv2 and CMIP. The practical guide to Network-Management Standards"*, Prentice-Hall, 1993.