Effective Network Monitoring Using Mobile Agents

Vipan Arora and Dinesh Kumar

Government Polytechnic College for Girls, Jalandhar  
1DAV institute of Engg & Technology, Jalandhar  
E-mail : vipan.arora@gmail.com

ABSTRACT

Fast growth of computer Networks require efficient network management and monitoring tools for better utilization of resources. Traditional centralized Network Management Systems (NMS) are not adapted to wide spectrum of heterogeneous network installations and configurations. Major problem areas are heterogeneity, complex topologies, scalability, limited bandwidth constraints etc. Present research activity is centered around providing distributed network management oriented intelligence to different network components. Mobile Agent (MA) paradigm seems to offer a good solution. Most network devices employ Simple Network Management Protocol (SNMP) agents for network management based on traditional centralized client/server architecture. In this paper, we propose a Mobile Agent Framework (MAF) integrated with SNMP. It provides a mobile network manager with Mobile Agent Generator (MAG) functionality. Generated MA can aggregate SNMP data, semantically compress and intelligently filter it. We present a novel approach to write string based network health functions and SNMP table filtering expression. A new concept of customized reliable traps based on health functions is introduced. Implemented MAF is also strengthened to handle and transfer large SNMP tables.

Keywords: Network management, monitoring, SNMP, mobile agent, table filtering, string based expressions, itinerary partitioning, health functions.

1. INTRODUCTION

Traditional network management (NM) protocols have been based on centralized client/server approach, which is quite unsuitable in modern computer networks. Two major protocols in use are Simple Network Management Protocol (SNMP) [1] and Common Management Information Protocol (CMIP) [2]. SNMP protocol assumes network to be collection of managed objects (hosts, routers, switches, bridges etc.) and each managed object runs a daemon (called SNMP Agent) that continuously updates Management Information Base (MIB) [3] with network management statistics. MIB is stored locally in managed object. Network Management System (NMS) employs a Manager application (centrally placed with network administrator).

SNMP agents respond to the queries (GETxxx) made by manager. But frequent monitoring (query-response) of network state parameters (MIB variables) leads to polling, that typically involves massive transfers of management data causing considerable strain on network throughput and creating a processing bottleneck at manager. Here mobile Agents can prove to be an effective solution [4][5][6]. The MA can go to desired managed object and interact (poll) with SNMP agent locally, thus saving on costly WAN bandwidth.

Traditional network management systems are based purely on SNMP client/server architecture.
Various research Mobile Agent Frameworks (MAF) integrated with SNMP have been proposed [7][8][9][10][11] and implemented in network management area. We call such MAF as 2nd generation NM hybrid models. A brief review of such models is made to formulate basic NM concepts and introduce research voids.

In this paper we propose another NM hybrid model. We introduce and implement several novel approaches in this model such as 3-dimensional network state, string based expressions, customized reliable traps based on health functions, global filtering, enhanced SNMP table handling and itinerary partitioning strategy [12]. This model has been implemented in our network monitoring application (NetMonitor) developed using IBM’s Aglet SDK [13], AdventNet SNMP API [14].

2. REVIEW OF DISTRIBUTED NETWORK MANAGEMENT USING HYBRID MODELS

Zapf’s [7] hybrid model NetDoctor is based on Asynchronous MEssage Transfer Agent System (AMETAS) platform. It depicts network state using health functions. MAF includes Mobile Agent Generator (MAG), called, User Adapter, generates Health Agents that migrate to host, compute health functions in a polling mode, compare it with a threshold set, return to User Adapter in the event of exceeding threshold or completing polling time. In this model health functions are based on mobile code repository approach, thereby complex functions cannot be generated or user has to provide code for these. This model lacks SNMP table handling. Filtering of data is neglected altogether.

Antonio Pualiafito’s [8] hybrid model, based on Mobile Agent Platform (MAP), also uses health functions to monitor network state. It introduces customized health functions by means of daemon agents. The messenger agent visits nodes to collect data computed by daemon agents. It provides abstract classes to customize health functions, thereby putting the burden on user/network manager to write the code for health agents. This model as well as several other hybrid models [10][11] have also the similar constraints.

D. Gavalas’s [9] hybrid model provides better SNMP table handling and filtering. It also provides abstract java methods for writing filtering expressions. It also uses mobile code repository approach that lacks scalability. Complex filters cannot be realized. Network manager has to write the mobile code himself for custom filters or get contended by a handful of ready-made filters in the mobile code repository provided along with the application.

We propose a String based expression building approach to write customized health functions and filters. Now network manager can have unlimited filters. Customization involves just writing an arithmetic/Boolean expression in the GUI. In addition, our platform uses itinerary partitioning [12] approach to alleviate bloated state problem[12] to reduce network time response. The framework also extends the SNMP trap capability and makes it more reliable.

3. BASIC NETWORK MANAGEMENT HYBRID MODEL CONCEPTS

3.1 SNMP Data Retrieval operations

SNMP data transfer is based on User Datagram Protocol (UDP). UDP was used to keep transport layer overhead to minimum for small sized Protocol Data Units (PDU) and better network latency, but we lost reliability in the tradeoff. SNMP v1 supports two basic data retrieval operation, GET (Ref Fig 1(a)) and GETNEXT (Ref Fig 1(c)). SNMP v2 has added GETBULK operation (Ref Fig 1(d)). Multiple MIB-II objects, also called variables or Object Identifiers (OIDs) can be given in one GET request (multi-varbind feature) Ref Fig 1(b). Number of OIDs in multi-varbind request are limited by maximum PDU size supported by SNMP agent on managed object. GETBULK operation is equivalent to issuing multiple GETNEXT (governed by max-repetitions parameter).

SNMP GETNEXT is used to walk down the MIB-II tree or to access SNMP tables. GETBULK enables manager to receive large block of data efficiently by specifying a maximum number of
successive values to be returned (max-repetitions). But now manager has to guess a value for max-repetitions parameter. Using small numbers for this parameter results in too many SNMP PDU exchanges and using large number may cause (Ref Fig 1(d)) the SNMP GETBULK request to be rejected by SNMP agent since response PDU becomes too large (exceeds capability of SNMP agent, most agents has this limit as 64kB).

\[ \text{DiscardRate}(t) = \frac{\text{ipOutDiscards} + \text{ipOutNoRoutes} + \text{ipFragFails}}{\text{ipOutReqs} + \text{ipFwdDatagrams}} \times 100 \]

Fig. 2: Example of Health Function involving many MIB-II objects

DiscardRate(t) defines the percentage of IP output datagrams discarded over the total number of datagrams sent as recorded by managed object at time ‘t’, (when the SNMP response was generated). We see that evaluating this function requires 5 MIB variables to be queried. 1st option is to send 5 SNMP GET queries and get 5 responses (Ref Fig 1(a)). 2nd option is combine these 5 MIB variables in multi-varbind request (Ref Fig 1(b)) to generate one SNMP GET PDU, but the response PDU gets bigger (may get rejected by SNMP agent), since now it contains 5 OIDs and their values. 3rd option is to compute this function locally on managed object and return single value DiscardRate(t) to manager. The reduction in bandwidth becomes significant when multiple samples of this health function are required. MA having health function capability is thus able to perform semantic compression of managed information. Zapf [7] calles such agents as Health agents.

3.3 SNMP Traps

The SNMP standard defines seven traps that can be generated by SNMPv1 agents. Six of these traps are “generic” traps and the seventh trap is enterprise-specific. The enterprise-specific trap is used by the private organizations to define their device-specific traps. The generic traps are fixed and cannot be defined. On the other hand, it is possible to define multiple enterprise-specific traps. The six generic trap types defined for SNMPv1 agents are as follows.

- coldStart
- warmStart
- linkDown
- linkUp
- authenticationFailure
- egpNeighborLoss

The traps also use UDP and thus are inherently not reliable (may get lost). Secondly customized traps can not be generated e.g. To generate a trap “when 60% or more links(interfaces) get down” at one managed object, is not feasible with SNMP traps.
4. PROPOSED NETWORK MANAGEMENT CONCEPTS

4.1 3-Dimensional Network State
We define 3D-network state as a set of values of plain MIB variables, values of custom network health functions and custom filters e.g. SNMP table filters and general filters (Ref Fig. 3) e.g.

\[
\text{NS}(t) = \{ \text{sysName, ifNumber} \ldots \},
\]

\[
[\text{DiscardRate}(t), \text{utilization}(t) \ldots],
\]

\[
[\text{ifTable-Filter(up)}, \text{ifTable-Filter(lowSpeed)}, \text{tcpConnTable}(listen) \ldots] \}
\]

\[
\text{Network State } \{\text{OID, HF, Filter}\}
\]

\[
\text{Fig. 3: Envisaged 3-D Network State}
\]

NS(t) represents network state at time (t) with 1st dimension as set of Plain OIDs = \{sys Name, if Number\ldots\}, 2nd dimension as a set of health functions={DiscardRate(t), utilization(t)\ldots}, 3rd dimension as a set of filters={ifTable-Filter(up), ifTable-Filter(lowSpeed), tcpConnTable(listen)\ldots}. This enables a single MA to fetch all the required data. In hybrid models reviewed in para 2, multiple MA are required to be launched to achieve the same i.e. greater consumption of bandwidth and higher network latency.

4.2 Proposed Health Function Format and Reliable customized Traps
The proposed health function will have three attributes i.e. \{name, expression, isTrap\}. The name (type-String) is unique for a MA, expression (type-String) may comprise of valid OIDs, integer/float/string constants. It may evaluate to a number, string or a boolean value. We merge the concept of health functions with traps, isTrap, is a boolean attribute, indicating that this health function will generate a NetTrap agent, whenever expression evaluates to true (i.e. expression must be Boolean, if isTrap is set). The agent migration is TCP/IP based. Thus, we have more reliable traps than SNMP traps, which are UDP based.

Further health function can be literally anything, limited by network administrator’s imagination. e.g. expression in Fig. 4 will generate a trap if some managed host has more than 3 interfaces, and establishes a TCP connection on port 80 on any of the interface.

\[
\{ \text{tcpConnLocalPort==80} \&\& \text{tcpConnState==5} \&\& \text{ifNumber>3} \}
\]

\[
\text{Fig. 4: Customized Health Function based Trap}
\]

4.3 Proposed String based Expressions
We also introduce a string based approach to create complex expressions for health functions and filters. Conventional hybrid models (Ref para 2) used mobile code repository based approach (providing abstract classes, interfaces, methods etc.) for generating health functions. This limits creating a customized complex filter/health function. We integrate Jformula [15] expression parser with our MAF to enable string based expressions.

\[
\text{Fig. 5: Using JFormula for String Based Expression}
\]

All symbols in expressions are parsed, if they are valid OIDs. The valid OIDs are categorized as Long, String type. OID symbol in expression is replaced accordingly by making a SNMP GET query for evaluation of the expression.

4.4 Proposed Filter Format
The filters are used to select desired information while discarding unwanted information, thus reducing MA state size. We envisage filters to be of two categories
The filter format is taken from snmpSQL [16] SELECT command. The example Table filter in Fig. 6 will pick \{ifPhysAddress, ifDescr, ifType\} columns from table ifTable store in all the nodes that MA visits. The result will include only those interfaces, which have speed greater than 1500. The filtering expression (ifSpeed>1500) is applied after the table is sorted in ascending order based on ifPhysAddress. If more than 2 table rows satisfy filtering expression, then top two rows are picked. The final results i.e. no. of rows will be restricted to 50. The General filter will not have tableName, the OIDs given in select attribute must be scalar. Rest of the format and working remains the same. The filtering expression must be Boolean for both type of filters.

4.5 SNMP Table Handling Improvements

An SNMP table can be defined as an ordered collection of objects consisting of zero or more rows. Each row may contain one or more objects. Each object in a table is identified using the table index. A table can have a single index or multiple indices. A scalar variable has a single instance and is identified by ".0" as suffix in its OID. On the other hand, a table object or the columnar variable can have one or more instances and is identified by its index value. To identify a specific columnar variable, the index of the row has to be appended to its OID.

For example, consider tcpConnTable. It has four indices namely tcpConnLocalAddress, tcpConnLocalPort, tcpConnRemAddress, and tcpConnRemPort (Ref Fig.7). To get the value of the column tcpConnState for the last row, we have to query with the OID tcpConnState.192.168.1.78.1156.192.168.4.144.80 where (192.168.1.78, 1156, 192.168.4.144,80) are the value of indices. Since SNMP table size is unknown, multiple GETNEXT, or multiple GETBULK requests are used. We first examine the version of the SNMP Agent on managed host, by sending 3 SNMP PDUs (set to different versions) to the desired managed host. We will get reply for the correct version, rest will time out. If SNMP Agent is version 2c and above, we use GETBULK for reduction in bandwidth consumption.

Though AdventNet SNMP API [14] provides high level API constructs for table handling such as getColumn(), getRow() etc. through java beans, but these are not efficient in terms of bandwidth consumption and network latency. We use low-level API [14] constructs to generate SNMP PDUs to get the table. The table is converted to 2D array and filters are applied to trim rows/columns.

4.6 Proposed Itinerary Partitioning Strategy

The user defines initial Itinerary I for MA as \{N1, N2, N3, ..., Ni, Ni+1, ..., Nn\}. Here we define a partition factor p (p>0) to control itinerary partitioning. MA is launched from home Node H to 1st node in itinerary i.e. N1. On arrival at node Ni mobile agent examines itinerary I, if number of nodes in I i.e. |I| > p, then itinerary is partitioned into I0={N1, N2, ..., Np} and I1=I-I0={Np+1, Np+2, ..., Nn}. After this MA is cloned. The original MA is called C0 and cloned MA is called C1. Now C0 is allotted itinerary I0 and C1 is allotted itinerary I1. The original aglet C0 is not cloned any more but C1 repeats this procedure on arrival to next node assuming the role of C0.
Further we notice that clones are launched from first node in partitioned itinerary (indicated by a kink in Fig. 8(b) and Fig. 8(c)) and not launched from home node. This also yields performance improvement, if initial itinerary has been given based on network topology. The cloning is done immediately on arrival at node, any SNMP polling is done, after the clone is prepared, so that it can be immediately dispatched to next node on priority. However SNMP polling and clone dispatching is done concurrently in two separate threads. In nutshell, itinerary $I$ is partitioned into $k = \frac{\lambda}{p}$ itineraries $I_0, I_1, \ldots, I_{k-1}$ where last itinerary $I_{k-1}$ may contain less than $p$ nodes and itinerary $I_i$ is taken care by clone $C_i$. In Fig 8, three cases of this strategy has been shown with $p=n$, $p=1$, $p=2$. The first two cases are extreme. The case-I (with itinerary $I_0=\{N_1, N_2, \ldots, N_7\}$) is being followed in conventional approaches, i.e. No Partitioning [6][7][8], whereas case-II reflects the extreme cloning to obtain immediate results. In case-II we have 6 itinerary parts i.e. $I_0=\{N_1\}, I_1=\{N_2\}, \ldots, I_6=\{N_7\}$. In case-III we have 4 itinerary parts i.e. $I_0=\{N_1, N_2\}, I_1=\{N_3, N_4\}, I_2=\{N_5, N_6\}, I_3=\{N_7\}$. The intermediate values of $p$ i.e. $1 < p < n$ reflect intermediate response time cases and may be chosen as per application demands.

5.7 Network Monitoring Architecture

The proposed network management concepts (Ref para 4) have been implemented and evaluated using a MA based approach integrated with SNMP using IBM Aglet SDK [13] and AdventNet’s SNMP API [14]. The proposed architecture (NetMonitor Ref Fig. 9) is based on three aglets, termed as NetAdmin NetMonitor and NetTrap aglet.
3D network state data i.e. data for plain OIDs, OIDS in health function expression and filter expressions. After evaluating expressions it applies filters. If for some health function, isTrap attribute, is set, it sets up NetTrap aglet and dispatches it to home node. MA compresses data before migration to next node in itinerary, (if none, then migrates to home). Upon coming to home it generates result Message and sends it to NetAdmin aglet for processing network state and to launch a UI for displaying the results.

It records various timings and evaluation parameters such as on arrival, before dispatch, preprocessing time, SNMP polling time, cloning time, state size accumulated etc. for studying effectiveness of the proposed approach.

(c) NetTrap Aglet it is very small aglet, launched by NetMonitor, in the event of some trap being fired (isTrap health function expression evaluates to true). It generates trap Message, upon reaching home node and delivers it to NetAdmin aglet. The NetAdmin aglet notifies the received trap to manager via a UI. However, automatic network recovery decisions can be made and implemented using MA based approach, but are not catered for in our MAF.

5.8 Performance Evaluation

The implementation has been tested on network shown in Fig 11. The N1, N5 nodes have Linux OS and remaining are based on WinXp. All nodes have Intel PIV 2.4GHz, 512MB RAM and are connected using 100Mbps Ethernet.

![Fig. 10. Test Bed Network for NetMonitor](image)

Fig. 10: Test Bed Network for NetMonitor

The NetAdmin aglet was configured to launch NetMonitor aglet to fetch data for plain MIB variables sysDescr, sysUpTime, sysName, ifNumber, ipFragFails, icmpInMsgs, icmpInErrors, tcpMaxConn, tcpInErrs, tcpOutRsts. The Table filters

![Fig. 11: NetAdmin Aglet UI](image)
Effective Network Monitoring Using Mobile Agents

183

on tables ifTable, ipAddrTable, ipRouteTable, tcpConnTable, udpTable were also added. Health function $DiscardRate(t)$ (Ref para2) was also monitored. The test runs were made first by varying no. of snapshots/samples of these variables from 1 to 6 as well as varying partition factor $p$ from 1 to 5 and with/without using table filters. The results with $p=5$ (i.e. No Itinerary Partitioning, and No Filters/Health Functions) have been shown in Fig. 12(a). We observe that as MA migrates from home node to N5, its state size increases that affects migration time also.

If we compare state size increase in Fig 12(a) and 12(b), the semantic compression of state is visible. The results in Fig 12(b) were obtained when we used table filters and health functions. The data aggregated at each node reduces to $1/3^{rd}$ approximately. The reduction in state size depends upon how much filtering of data has taken place.

The effect of itinerary partition factor is observed from Fig. 13. We obtain least network response time with $p=1$, where SNMP polling takes place simultaneously on all nodes in the itinerary. As we increase $p$ the response time increases and is limited by $p=n$ i.e. conventional itinerary approach.

6. CONCLUSION

In this paper, we have discussed research approaches in the area of network management and monitoring that employ hybrid models i.e. using mobile agent technology integrated with SNMP. We proposed some new concepts like 3-dimensional network state, string based expressions, customized reliable traps based on health functions, global filtering, enhanced SNMP table handling and itinerary partitioning strategy. These concepts have been implemented in our MAF (NetMonitor) and performance improvements analyzed. String based expression increases scalability w.r.t. creating customized health functions and filters in a most effective manner. Itinerary partitioning approach reduces network response time.

Our future work involves discovering network topology for efficient and intelligent planning of itinerary to achieve better network response time at NMS.

REFERENCES

15. JFormula Expression parser API for java http://www.japisoft.com/formula