

SCALABLE POLICY MANAGEMENT FOR AD HOC NETWORKS¹

Ritu Chadha*, Yuu-Heng Cheng*, Jason Chiang*, Gary Levin*, Shih-Wei Li*, Alexander Poylisher*,
Lorraine LaVergne[†], Scott Newman[†]

*Telcordia Technologies, One Telcordia Drive, Piscataway NJ 08854, USA.

[†]Network Management Division, CERDEC, U.S. Army, Fort Monmouth, NJ 07703, USA.

Contact Author: Ritu Chadha; chadha@research.telcordia.com; Tel: +1-732-699-2987; Fax: +1-732-336-7025.

ABSTRACT

The characteristics of mobile ad hoc networks (commonly called MANETs) are sufficiently different from commercial wireline networks to have generated a great deal of work in alternate management paradigms. The task of managing MANETs involves frequent reconfiguration, due to node mobility and consequently dynamically changing network topology. Network nodes often have limited battery power and storage capacity, and wireless radio link capacity and quality varies dynamically based on environmental conditions (weather, terrain, foliage, etc.). These differences have resulted in a need for paradigms particularly suited for managing MANETs. In this paper, we describe a distributed, hierarchical management system that implements policy-based control of a MANET. Policies are used to enable flexible composition of diverse management actions based on network conditions and external events. A policy conflict detection mechanism based on event calculus is used to detect certain types of policy conflicts. The rapidly changing network topology and link characteristics typical of a MANET are managed by an adaptive, self-forming hierarchy of policy agents that implement the appropriate management actions based on their management role. The system described in this paper has been prototyped and demonstrated in a laboratory environment.

1 INTRODUCTION

Traditional network management functionality is divided into the following components, commonly known as FCAPS: Fault, Configuration, Accounting, Performance, and Security Management. Typical network management solutions provide vertically stove-piped products that address one or two of the above areas, with, at best, some loose coupling between the products. The fundamental problem with this approach is the inability to:

- Tie together the operation of these systems; and
- To do so in a way that can be changed as needed depending on management goals.

The first problem above needs to be addressed by closing the feedback loop between network status and network re-configuration, as has been discussed in previous work [4]. However, more importantly, the second problem indicates that the interactions between network status gathering and network re-configuration need to be expressible in a flexible, user-friendly manner, so that these interactions can be customized as needed based on possible changing management goals. This problem is accentuated in military scenarios, where missions are typically short-lived (72 hours or less) and the requirements of each mission require that the network and its mode of management be tailored to the needs of the mission. For example, certain missions in enemy territory require stronger encryption for over the air communications than missions in friendly territory; the nature of communications in different missions may dictate different bandwidth allocations for QoS traffic classes for each mission; certain missions may be particularly sensitive to jamming and interception; and so on.

In order to tie together the operations of FCAPS (Fault, Configuration, Accounting, Performance, and Security Management), we add another component to FCAPS: the *Policy Management* component, used to provide the glue between FCAPS functions. The Policy Management component is used to:

- Initiate the operations of FCAPS components, by kicking them off as needed;
- Control the operations of FCAPS components, by conditionally altering their behavior;
- Tie the operations of one FCAPS component to another, by capturing the desired interactions between these components;
- Allow the specification of all of the above in a user-friendly manner via policies.

As mentioned above, similar work has been done in PECAN [4], where the first three bullets above were addressed. What was missing in PECAN was a flexible

¹ The research reported in this document/presentation was performed in connection with contract number DAAD19-01-C-0062 with the U.S. Army Research Laboratory. The views and conclusions contained in this document are those of the authors and should not be interpreted as presenting the official policies or position, either expressed or implied, of the U.S. Army Research Laboratory, or the U.S. Government unless so designated by other authorized documents. Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

framework for adding FCAPS functionality into the management system without having to re-code system interactions; and the ability to distribute the PECAN system functionality across a large network. The first of these shortcomings was addressed in the DRAMA policy management solution for mobile ad hoc networks [3] (referred to here as DRAMA Release 1.0), where we described a flexible agent-based framework that allowed insertion of FCAPS actions that could be driven via policies. The DRAMA 1.0 system also provided the ability to distribute policy agents across a mobile ad hoc network and provided a fixed, two-tier management hierarchy that allowed a *Global Policy Agent* (GPA) to control the operation of the network by distributing policies to *Local Policy Agents* (LPAs), which in turn each managed a single network node. However, the scalability of this system was limited by the fact that it was limited to a fixed, statically defined two-level hierarchy. Additionally, as mentioned above, every managed node was required to run a Policy Agent. This posed a problem because typical COTS network elements, such as commercial routers, cannot host arbitrary third-party network management software. Another limitation was that the operation of the system was closely coupled with the MOSAIC network [13], and depended on several MOSAIC protocols to perform some of its functions, including policy distribution. This was a serious limitation as MOSAIC protocols are not compatible with existing commercial and military network elements. Finally, the system did not perform any policy conflict detection, thereby allowing definition of policy rules that could not be enforced due to undetected conflicts with other rules or incompleteness in the defined set of rules.

In this paper, we describe the following DRAMA Release 2.0 enhancements to earlier DRAMA functionality:

- A self-forming, self-maintaining, multi-level hierarchy of policy agents;
- Policy conflict resolution based on event calculus; and
- An adaptive communications infrastructure.

1.1 Related Work

The characteristics of mobile ad hoc networks (commonly called *MANETs*) have resulted in a need for paradigms particularly suited for managing MANETs. As an example, although SNMP is the management protocol of choice for monitoring the vast majority of network elements available on the market today in wireline networks, alternative protocols have sprung up that are specifically designed for mobile ad hoc networks. ANMP (Ad hoc Network Management Protocol) [12] is a management protocol that is compatible with SNMPv3, and uses hierarchical clustering of nodes to reduce the number of messages exchanged between the manager and the agents. An-

other function of a network management protocol is to present the topology of the network to the network manager. In a wireline network, this causes minimal network overhead as network topology changes very infrequently; however, in a MANET, the extremely dynamic network topology results in a high volume of network management traffic. When this is coupled with the habitually scarce bandwidth and computing resources in MANETs, the result is significant network burden for reporting network topology to the manager. We use a hierarchical management system organization as well as dynamically adjustable reporting intervals to minimize the impact of reporting management information such as topology details across the network.

Policy-based networking [5] is a powerful approach to automating network management, as evidenced by the numerous industry efforts in this area dealing with diverse networking domains, e.g., network management [2], quality of service control [7], security [11], etc. A policy implies a pre-determined action pattern that is repeated by an entity whenever certain system conditions appear. The first “policy-based” applications in the area of computing were in the expert systems area, where “if-then” rules were programmed to determine the decisions of computer software in a specific knowledge domain. More recently, in the area of distributed systems management, work in [8] outlined an approach to providing automated support for analysis of policy hierarchies for the management of very large distributed systems. The reason for using policy hierarchies was to enable specification of high-level policies and generation of lower-level policies.

In [4], we described a policy-based management system called PECAN that was used to manage MPLS networks. The goal was to increase the level of automation of current management systems by automating the interactions between FCAPS components. This was accomplished by creating a feedback loop between fault/performance monitoring and configuration management, and by specifying policies that regulated how the system should be reconfigured in response to various network events. PECAN provided capabilities for controlling admission of traffic into an MPLS network based on the QoS guarantees required; assigning traffic flows to MPLS traffic engineered paths; and for feeding information network fault/performance monitoring to an MPLS traffic engineering component in order to reconfigure the network to alleviate the effects of any observed problems.

In [3], we described a policy-based network management system tailored to managing mobile ad hoc networks. The high-level architecture of this system is shown in Figure 1. As shown here, a collection of Policy Agents manage all the nodes in the mobile ad hoc network. At the highest

level, the *Global Policy Agent*, or GPA, manages multiple *Domain Policy Agents*, or DPAs. A DPA can manage multiple DPAs or *Local Policy Agents* (LPAs). An LPA manages a node. LPAs perform local policy-controlled configuration, monitoring, filtering, aggregation, and reporting, thus reducing management bandwidth overhead. Policies are disseminated from the GPA to DPAs to LPAs, or from DPAs to LPAs. Policy Agents react to network status changes on various levels (globally, locally, domain-wide) by automatically reconfiguring the network as needed to deal with fault and performance problems. In this architecture, any node can dynamically take over the functionality of another node to ensure survivability. A flexible agent infrastructure allows dynamic insertion of new management functionality. The implementation described in [3] allowed for a statically-defined two-level hierarchy of policy agents.

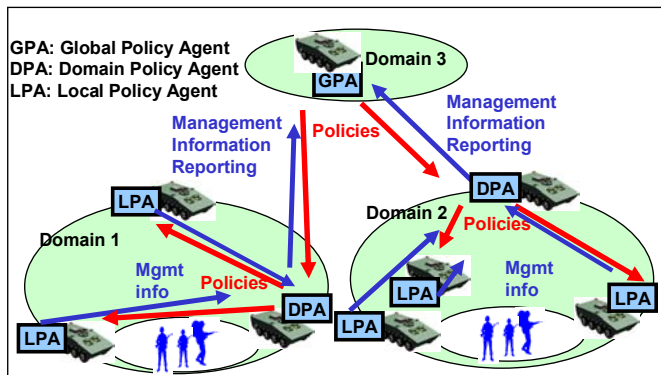


Figure 1. DRAMA Policy Agents

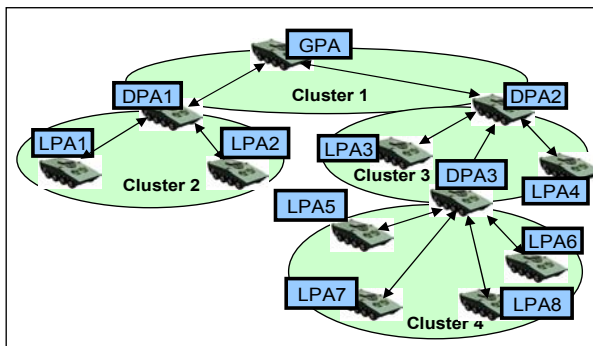


Figure 2. Sample Management Hierarchy

1.2 Outline

This paper is organized as follows. The next section provides an overview of the new features introduced in DRAMA 2.0. These include a self-forming hierarchy of policy agents; a conflict resolution mechanism; and an adaptive communications infrastructure that allows flexible adaptation to an ad hoc networking environment. Section 3 describes a laboratory implementation, and Section 4 provides a summary of future work.

2 SYSTEM OVERVIEW

The previous section included a brief overview of the previous release of DRAMA (Release 1.0). In this section, we describe the new features incorporated into DRAMA Release 2.0, which were briefly enumerated in the previous section.

2.1 Self-forming, self-maintaining, multi-level hierarchy of policy agents

The design philosophy of DRAMA is to create a set of intelligent, self-aware Policy Agents to manage the network and allow them to coordinate their actions without human intervention. These Policy Agents determine their roles — GPA, DPA, and LPA — in an autonomous manner in order to form a hierarchy (see Figure 1 for a sample management hierarchy). Policy Agents perform management actions based on their roles. If their roles change, they adjust their behaviors accordingly. Such a multi-tier hierarchical structure is essential for the scalability of the management system. The system can scale to the size of the network by expanding or shrinking the number of tiers in the hierarchy, and appropriately assigning management functionalities to nodes at different tiers in the hierarchy. One way to form a hierarchical structure in such management systems is to associate it with the human reporting hierarchy so that policies can be formulated to serve the management needs originating from the organization of human administration.

Given that a multi-tier management hierarchy is required for scaling a distributed management system, it is critical for this hierarchy to *systematically and autonomously self-form*, especially if the target networking environment is dynamic by nature. The reason for this is that nodes may disappear or become unreachable, thus requiring frequent re-organization of this management hierarchy. The basic idea for formation and maintenance of the management hierarchy is to form “clusters” and then link clusters into a tree. The concept of clusters is not new to MANETs, and has been explored in depth for the purposes of routing [10]. Many of the concepts of clustering from [10] are used here, including building and restructuring a clustering hierarchy. The important difference to note here is that we use clustering for the purpose of network management, rather than for routing. The concept has also been used for MANET management in [12], where a hierarchy of three levels is used for management. In contrast to [12], we allow a management hierarchy of arbitrary depth, since limiting the depth of the hierarchy intrinsically limits the scalability of the system. In DRAMA, a cluster includes a single cluster leader and a set of children, known as *cluster associates* or *cluster members*. Leaders of several clusters can form another cluster. A sample set of clusters is shown

in Figure 2.

2.1.1 Initial formation of management hierarchy

The initial formation of the previously described management hierarchy is achieved by advance planning. A planned management hierarchy is loaded into a policy naming server on the GPA node when the system is launched (note that the GPA is the root of the management hierarchy). In this architecture, we assume that the policy naming server is replicated at every policy agent; its function is to distribute and store data about the management hierarchy. The policy naming server on the GPA distributes the naming data, which essentially includes <child, parent> pairs, to all children under the GPA. Nodes receiving naming data store the data in their local store, and distribute the data to all their children, if any.

During the initialization phase on each node, a Policy Agent uses the policy naming service to find out whether it has a parent in the predefined hierarchy. If it has no parent, it assigns itself the role of GPA; if it has a parent, it checks whether it has any children to determine whether its initial role should be a DPA or an LPA. In case no naming information is available from the naming service, which could happen if the distribution of naming data by the GPA is unsuccessful due to intermittent connectivity, a Policy Agent will assign itself the role of GPA. This could result in more than one GPA for the network; this is handled via a merge after network connectivity is restored, as described below in Section 2.1.3.

2.1.2 Cluster maintenance

After the initial formation of the management hierarchy, it is necessary to maintain this hierarchy for distributing policies and collecting management information. Cluster maintenance is achieved as follows. First, the leader of a cluster periodically sends heartbeat messages to all its associates. These heartbeat messages contain cluster information such as cluster leader, cluster settings, and so on. A cluster associate acknowledges the receipt of heartbeat messages from its leader in order to maintain its membership in the cluster. If a cluster associate misses a number of consecutive announcement messages from its leader, it assumes that the leader no longer exists and tries to join the cluster of an ancestor of its current parent. If no ancestor can be reached, it changes its role to become a GPA. Further, if a cluster becomes too big (where “too big” is a configurable metric), it splits into two or more clusters.

For cluster signaling, two types of messages are used. When a node wants to join a cluster (when two clusters are merging), it sends a “join request” message to the cluster leader. If the request is granted, the leader sends a “join granted” message to the child and records the node as its child when an acknowledgement message is received.

When a cluster leader splits its cluster into two or more clusters, it appoints one of its associates as the leader for a group of associates in its cluster. The new leader remains a child in the current cluster, while the associates to be split out join the new cluster by signaling their new leader.

2.1.3 Management hierarchy maintenance

To exercise global control of a policy system, it is crucial to keep one single management hierarchy in the system as far as possible, so that policies can be distributed in a top-down fashion and information can be collected from the bottom up. This is done as follows:

- When a domain associate detects the existence of another GPA, it notifies its own GPA;
- When two GPAs detect each other, they negotiate with each other to merge into one hierarchy.

When a node receives a heartbeat message from a GPA other than its own, it forwards this message to its GPA. When a GPA learns the existence of another GPA, it initiates a negotiation session with the other GPA. The other GPA signals the session initiator its decision to remain a GPA or to step down and become a child of the session initiator. Once the session initiator acknowledges this decision, the two hierarchies merge and one of the GPAs steps down to become a child of the other. The two hierarchies also merge their contents to complete the merge process.

2.2 Policy Conflict Resolution

Policy conflict resolution was introduced in DRAMA Release 2.0 for monitoring, aggregation, and reporting policies only (see [3] for a detailed description of these types of policies). Typically, the term *policy conflict* refers to undesirable consequences of combined execution of actions prescribed by several policies. Such consequences ultimately include concurrent setting of the same variable by two different actions to two different values, and the resulting system behavior. Other forms of conflict include *incompleteness* of the policy set, wherein, for instance, the condition part of a policy rule includes a variable whose value is supposed to be produced as a result of another rule’s action, but such a policy does not exist. Other constraints, such as those based on the freshness of data explicitly set in policy conditions or implicitly in the policy type, can also be violated, leading to *inconsistent* policies.

An actual policy conflict refers to the conflict that *will* occur at some point in time, given the current set of policy rules, with probability 1. *Possible* conflict refers to conflict that *may* occur at some point in time, given some run-time conditions. There are three possible approaches to the *time* of conflict resolution: (a) resolve all conflicts at compile time; (b) resolve only actual conflicts at compile time and defer resolution of possible conflicts until the time of ac-

tual occurrence; and (c) resolve all conflicts at the time of actual occurrence. The approaches differ in their performance implications. Resolving possible conflicts at compile-time requires expensive what-if analysis. On the other hand, actual conflict resolution at run-time may affect system responsiveness. In DRAMA, we assume that possible conflicts will not occur very often, and avoid the expensive compile-time resolution for all possibilities. On the other hand, we would like to detect and resolve some easily identifiable conflicts before run time. Therefore, we choose the so called *balanced* approach (b) above as the most flexible, and implement only actual conflict resolution in DRAMA 2.0.

Conflict resolution is a two-step process. First, conflicting policy rules are *detected*. Next, an automated or manual *resolution* may be suggested by the conflict resolver. Resolution includes modification, deletion, or addition of rules to eliminate conflict. Automated resolution requires explicit resolution rules to be available to the conflict resolver. Such rules may, for instance, use rule precedence relations to decide which rule's actions should be executed. The DRAMA 2.0 implementation is limited to conflict *detection*, with subsequent manual resolution.

Following the approach developed in [1] and detailed in [9], we use event calculus (EC) as a formal language to represent DRAMA policies and conflicts, and detect actual conflicts using abductive reasoning. As shown below, it is possible to identify a set of generic constraints on monitoring, aggregation and reporting policies, applicable to any such policy expressed in the conventional event-condition-action notation. It has been shown [1] how a conventional high-level policy language can be automatically translated into EC; we use a similar approach for DRAMA policy rules. The domain-specific constraints, similar to those identified below, are expressed in EC. Once a combined representation is obtained, actual conflict detection is performed with an abductive reasoning procedure.

If a conflict is detected when a new policy is activated, or when an active rule is deleted or modified with the GUI or a command-line interface, the user is given the appropriate diagnostic feedback and is required to modify one or more rules until actual conflicts are eliminated. Below we analyze some policy conflict types that can arise in monitoring, aggregation and reporting policies [3].

2.2.1 Actual conflict types for monitoring policies

A monitoring policy rule can be specified as follows:

- **Event:** time event, arriving with periodicity T_m .
- **Condition:** none.
- **Action:** obtain current value of variable v , store in DB.

The following conflict types can be identified for monitor-

ing policies:

1. *Identical rules:* rules that differ only by rule name and/or the name of the DB object in which v is stored.
2. *Redundant rules:* a rule to monitor v with periodicity T_m and a rule to monitor v with periodicity nT_m , where n is any natural number.
3. *Data overwriting rules:* two rules to monitor v with periodicities T_m and T_k , where $m > k$ and $m \bmod k \neq 0$, assuming v is stored as a scalar. If v is stored as a vector indexed by periodicity T , this will not be a conflict.

2.2.2 Actual conflict types for aggregation policies

A *spatial* aggregation policy rule is specified as follows:

- **Event:** time event, arriving with periodicity T_a .
- **Condition:** a Boolean expression on multiple variables $\{v_i\}$, whose current values are available in the DB.
- **Action:** if condition evaluates to True, compute the value of an expression on one or more variables $\{v_j\}$, whose values are assumed to be available in the database, and store that value in the database.

In spatial aggregation, the variables v_k are treated as scalars. In addition to the conflict types described for monitoring policies above, the following types can be identified for spatial aggregation policies:

1. *Non-existent rules:* there exists v_k that is assumed to be refreshed by a periodic policy (i.e., a monitoring or aggregation policy), but there is no corresponding policy. Identification of this conflict type depends on the existence of an attribute for each database object that identifies it as the output of a policy.
2. *Incompatible periodicity:* for any v_k that is assumed to be refreshed by a periodic policy (i.e., a monitoring or aggregation policy), the following two constraints must hold: $T_k \leq T_a$ and $T_a \bmod T_k = 0$. Any violation of these constraints constitutes a conflict.
3. *Incompatible phase:* the maximum difference Δt_s between any two *start* times for any two policies that refresh the value of any v_k used in either the condition or the action part of an aggregation rule, exceeds a pre-defined value Δt_{s-max} . The latter is a configurable system-wide parameter.

A *temporal* aggregation policy rule is specified as follows:

- **Event:** time event, arriving with periodicity T_a .
- **Condition:** a Boolean expression on multiple frames of the variable v_i , whose values (indexed by time) are assumed to be available in the database.
- **Action:** if condition evaluates to True, compute the value of an expression on multiple frames of v_j , whose values are assumed to be available in the database, and store that value in the database.

In temporal aggregation, the variable v_i is treated as a vector. The following conflict types can be identified for temporal aggregation policies, in addition to those described for monitoring policies:

1. *Non-existent rules*: v_i is assumed to be refreshed by a periodic policy (i.e., a monitoring or aggregation policy), but there is no corresponding policy. Identification of this conflict type depends on the existence of an attribute for each database object that identifies it as the output of a policy.
2. *Incompatible periodicity*: v_i is assumed to be refreshed by a periodic policy (i.e., a monitoring or aggregation policy), and the following constraint must hold: $T_i \leq T_a$. Any violation of this constraint is a conflict.

2.2.3 Actual conflict types for reporting policies

A reporting rule can be specified as follows:

- **Event**: time event, arriving with periodicity T_r .
- **Condition**: a Boolean expression on multiple variables $\{v_j\}$, whose current values available in the DB.
- **Action**: if condition evaluates to True, report the value of a variable v_j , which is available in the database.

In reporting, the variables v_k are treated as scalars. The following conflict types can be identified for reporting policies:

1. *Identical rules*: rules that differ only by rule name and/or the name of the database object in which v is stored
2. *Redundant rules*: a rule to report v with periodicity T_m and a rule to report v with periodicity nT_m , where n is any natural number.
3. *Incompatible periodicity*: for any v_k that is assumed to be refreshed by a periodic policy (i.e., a monitoring or aggregation policy), the following two constraints must hold: $T_k \leq T_r$ and $T_r \bmod T_k = 0$. Any violation of these constraints constitutes a conflict.
4. *Incompatible phase*: the maximum difference Δt_s between any two *start* times for any two policies that refresh the value of any v_k used in either the condition or the action part of a reporting rule, exceeds a predefined value Δt_{s-max} . The latter is a configurable system-wide parameter.

While the set of constraints identified above is used as an initial set and may be modified and extended in the future, such constraints are clearly generic to the monitoring domain and independent of the actual rules.

2.3 Adaptive Communications Infrastructure

One of the goals of DRAMA 2.0 is to demonstrate its ability to manage a variety of networks in an ad hoc networking environment. Target networks include those being de-

ployed in various military programs, such as the U.S. Army’s Future Combat Systems (FCS) program. However, the operations of a management system cannot be completely separated from the underlying network as management information needs to be transported using the underlying network infrastructure. Hence we have developed an adaptive communications infrastructure that enables us to minimize the impact of dealing with highly dynamic ad hoc networks. Our solution is to control the distribution mechanism in a coordinated, adaptive fashion via a communications stack that allows components in the stack to be replaced, activated, and deactivated easily. Each layer in this communications stack forms an overlay network with its peering layers. Each of these layers is discussed briefly below. For a detailed discussion of this communications stack, see [6].

Application Layer: Application components (such as the Policy Agent) need to communicate with their peers. All communications are supported by the Distribution Layer. Depending on the instructions specified in the parameters of the invoked distribution method, the Distribution Layer will distribute policy operations as instructed.

Distribution Layer: This layer shields the components on the Application Layer from having to deal with the dynamics of the underlying network. This layer is included in this architecture mainly because the bulk of messaging in the policy system falls under the category of point to multi-point communications. Session support is also provided to cope with the addressing issues resulting from node mobility and network disconnection.

Transport Layer: This layer directs communications over available network transport protocols such as TCP and UDP. It is the appropriate place to provide functions such as encryption, compression, and QoS configurations.

Bearers: This layer represents the actual networking services available to the Policy Communications stack; one of the purposes for DRAMA Release 2.0 is to be independent of the underlying networking technologies.

3 IMPLEMENTATION

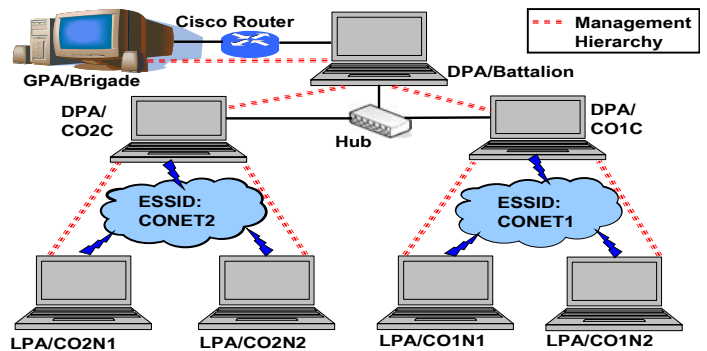


Figure 3. Demonstration Network

The functionality of DRAMA 2.0 was implemented and demonstrated in a laboratory environment in February 2005. The network setup is shown in Figure 3 above. The initial network consisted of a 4-level hierarchy of GPA, DPAs, and LPAs. The GPA (Brigade-level) and DPA at the Battalion level were connected via a Cisco router. The three DPAs were connected via an Ethernet hub. The initial management hierarchy is shown via a dashed line in Figure 3. Two 802.11b wireless networks with two LPAs and one DPA each formed the lowest levels of the management hierarchy.

Policies for monitoring and reporting node and interface status were distributed throughout the network. Every node was also configured with GPS monitoring and reporting policies that were activated as follows. Every node was instructed to monitor its GPS coordinates; however, a reporting policy was put in place to restrict GPS reporting to nodes that were within a certain pre-defined geographical region. The basis for this was that this region was supposed to represent an area of heightened enemy presence and therefore it was necessary for the commander at the GPA to track the location of all friendly forces within that region. Reporting of location information for forces outside that region was not a mission requirement and therefore bandwidth was conserved by triggering reporting via policies as described above. Additional scenarios demonstrated the use of policies for recovering from faults by installing policies that could automatically trigger corrective actions based on reported network events. As an example, faulty configuration of a router was detected via receipt of an SNMP trap (an interface was mistakenly shut down) and the suggested corrective action, namely to bring the interface back up, was automatically triggered. Conflict detection was demonstrated by creating a reporting policy without a corresponding monitoring policy; multiple monitoring policies for monitoring the same variable at different time intervals, etc. The conflict detection module detected these conflicts and provided diagnoses of the problem. Finally, automated management hierarchy maintenance was demonstrated by shutting down the interfaces of DPA/CO2C one by one, and observing the automated re-formation of the management hierarchy. When the network was thus partitioned, two management hierarchies, each with their own GPA, were formed. The interfaces were then restored and the original management hierarchy automatically re-formed.

4 FUTURE WORK

The emphasis of DRAMA 2.0 was directed at making the DRAMA system scalable, by enabling self-formation and maintenance of a management hierarchy, management of heterogeneous network elements, and by creating middleware for flexible and adaptable communications. Future

work will be directed at experiments that will use network performance data gathered by the system as input to control its own operation. The formation of the management hierarchy can be adapted based on network conditions; for example, link quality may dictate optimal cluster sizes; geographical data may influence cluster leader election; etc. Also, the choice of data distribution mechanism can be based on monitored network conditions; e.g. if the network is experiencing very high loss rates, the transport protocol for distributing management data can be automatically changed from TCP to reliable UDP. A very important component of this work will be a scalability study conducted via modeling and simulation to demonstrate scalability of up to 500 nodes. The system will be demonstrated in an outdoor environment at TRL-6 in Sept. 2005.

REFERENCES

1. A. K. Bandara, E. C. Lupu, A. Russo, "Using Event Calculus to Formalise Policy Specification and Analysis", Proc. of the 4th IEEE Intl. Workshop on Policies for Distributed Systems and Networks, June 2003.
2. R. Bhatia et al., "Policy Evaluation for Network Management", INFOCOM 2000.
3. R. Chadha et al., "Policy-Based Mobile Ad Hoc Network Management", 5th IEEE Intl. Workshop on Policies for Distributed Systems and Networks, June 2004.
4. R. Chadha et al., "PECAN: Policy-Enabled Configuration Across Networks", Proc. of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks, Como, Italy, June 2003.
5. R. Chadha, G. Lapiotis, S. Wright, "Policy-Based Networking", IEEE Network special issue, March/April 2002, Vol. 16 No. 2, guest editors.
6. C. Chiang et al., "AMS: An Adaptive Middleware System for Wireless Ad hoc Networks", IEEE MILCOM 2005, to appear.
7. A. Chiu, S. Civanlar, R. Rajan, "A Policy based approach for QoS on demand over the Internet", 8th Intl. Workshop on QoS (IWQoS), 2000.
8. J. Moffett, M. Sloman, "Policy Hierarchies for Distributed Systems Management", IEEE JSAC 11, 1993.
9. A. Russo, R. Miller, B. Nuseibeh, and J. Kramer, "An Abductive Approach for Analysing Event-Driven Requirements Specifications", Proc. of the 18th Int. Conf. on Logic Programming (ICLP), 2002.
10. M. Streenstrup, "Cluster-Based Networks", Ad Hoc Networking, Addison-Wesley 2001, ed. C.E. Perkins.
11. D. Trcek, "Security Policy Management for Networked Information Systems", NOMS 2000.
12. W. Chen, N. Jain, S. Singh, "ANMP: Ad hoc Network Management protocol", IEEE J. on Selected Areas in Comm, 17(8), 1999, 1506-1531.
13. K. Young et al., "Ad Hoc Mobility Protocol Suite for the MOSAIC ATD", MILCOM 2003.