POLICY-BASED MANAGEMENT OF THE FUTURE AIRBORNE NETWORK VIA PEER-TO-PEER NETWORKING

Elizabeth G. Idhaw, Lucas M. Lam, Dylan Pecelli, and Steven V. Pizzi

The MITRE Corporation
202 Burlington Road
Bedford, MA 01730

ABSTRACT

In this paper we present two prototype implementations of the policy-based network management framework, using peer-to-peer (P2P) protocols to improve policy distribution for the future IP-based Airborne Network.

The policy-based management framework defines an architecture that simplifies and distributes network device configuration commands and thereby streamlines deployment of a coherent network-wide configuration scheme. The management capabilities offered by policy-based management may enable consistent quality of service (QoS) throughout the future Airborne Network, which is particularly important because of its highly dynamic topology and bandwidth limitations. However, the current policy-based management applications that have been developed for fixed-infrastructure networks do not present an effective solution for the Airborne Network.

A more suitable approach for policy-based management of the future Airborne Network is a distributed system that can dynamically discover network devices. By providing distributed services and peer-discovery mechanisms, peer-to-peer networking appears to be an ideal candidate architecture for such a system. In JXTA, which is an open set of P2P protocols, there exists the potential to develop a more efficient policy-based network management application designed to meet the needs of the Airborne Network.

I INTRODUCTION

In this paper we present a prototype implementation of the policy-based network management framework, using peer-to-peer (P2P) protocols to improve policy distribution for the future IP-based Airborne Network. Policy-based network management [1]-[3] is a distribution methodology to provide a formal, high-level, adaptive, uniform management strategy across an entire network in order to enable consistent network behavior. The policy-based network management framework consists of the policy management tool, the policy repository, the policy decision point (PDP), and the policy enforcement points (PEPs). The policy management tool provides a graphical user interface for defining the policies, which are stored in the policy repository. The PDPs distribute the policies in the repository to the PEPs, which in turn implement them. Figure 1 illustrates the framework.

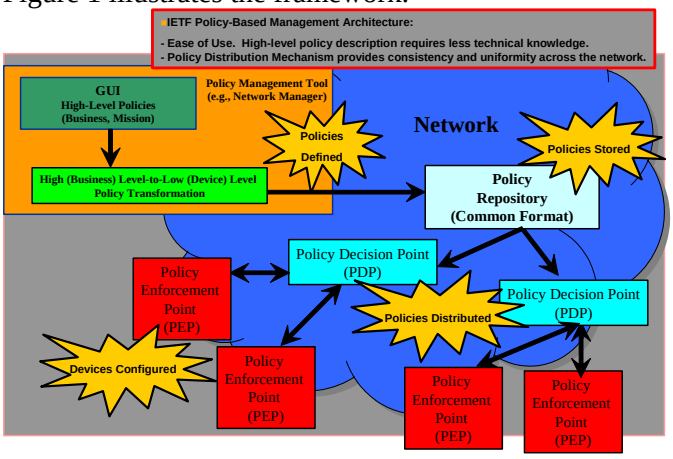

Figure 1. Policy-Based Management Framework.

Current research and development in both P2P and policy-based network management have yet to address the issues the Airborne Network will need to confront. In [4] P2P network management is explored as a means to improve current network management approaches. However, [4] focuses on fixed infrastructure networks and interdomain routing.

From the viewpoint of the Airborne Network, the key limitation with existing policy-based management systems today is their reliance on contemporary fixed terrestrial infrastructure networks. These management systems typically lack the capacity to adapt to changing network conditions and topologies. As a result these fixed-infrastructure-based approaches are an ineffective solution to providing consistent quality of service (QoS) for the dynamic environment of the Airborne Network.

A promising alternative to such systems can be found in Peer-to-Peer (P2P) networking. A P2P network is defined as a distributed network with all nodes sharing resources, such as processing power and storage capacity, some or all of which can be directly accessed by the other nodes in the network [11]. By sharing resources, the nodes in a P2P system, called peers, are able to provide particular services to one another, as opposed to the traditional client-server architecture, where each node plays only the client role or server role.

These peers cooperate to provide such services, distributing burdens to available resources throughout, and in so doing act as both client and server in accordance with immediate needs. Peers dynamically join and participate in the network, incrementally increasing its capacity, and are also free to leave the network without drastically diminishing the service provided. Distributing the task of providing services throughout all of the peers eliminates the danger of a single critical point of failure at a central service provider, providing more reliable service to all.

By providing distributed services and peer-discovery mechanisms, peer-to-peer networking appears to be a promising candidate architecture for the Airborne Network. In JXTA [5]-[10], which is an open set of P2P protocols, there exists the potential to develop a more efficient policy-based network management application which effectively addresses the needs of the Airborne Network.

II APPROACH

The Peer-to-Peer (P2P) networking approach, at least the one characterized by JXTA [5], appears to be a much more effective architecture on which to build a policy-based management application for the Airborne Network.

Again P2P networks provide a more distributed architecture, since each peer can communicate with any other peer to provide a service. By implementing P2P to create a distributed policy-based network management application for policy-based network management, it is not necessary for each PDP peer to communicate directly with a central policy repository. Instead the PDP peers can communicate with each other and propagate the repository information throughout the network. This approach provides a fully distributed architecture. Clearly, a central repository may still exist, but again it would be distributed to multiple PDP peers, so that not every PDP peer would need to communicate directly with the central repository itself to obtain or update the policies.

A benefit to distributing the policy repository is that the network traffic load can be shared through many routes and peers, instead of through select routes to central locations. This avoids single points of failure and better utilizes the available network bandwidth, thereby minimizing congestion [6]. In contrast, contemporary client-server systems focus these burdens on and around server nodes and leave no recourse when conditions cause servers to become unreachable. Such would be the case with a central policy repository. And with the dynamic nature of the Airborne Network, the central repository could often become unreachable.

Another feature that P2P protocols provide is a discovery mechanism. This discovery mechanism allows each peer to dynamically discover the other peers within the network. We believe discovery to be an extremely useful feature for the Airborne Network, since we expect there to be situations where network nodes will be entering and exiting, thus creating a continuously changing topology. With this changing topology, it may be less efficient to structure a policy-based network management application based on predefined PDPs and PEPs. Therefore, discovery of both PEPs and PDPs, as opposed to having them statically defined, could provide a more robust policy-based network management application for the Airborne Network.

Typically, for protocols to provide a discovery mechanism, they must use periodic messages, which may not scale well in very dynamic environments. However, scalability, a typical issue with most peer-to-peer networks, may not be an issue here, since we expect the number of peers to be relatively small and the frequency of policy distributions to be limited. Further research is necessary to clarify scalability concerns for the Airborne Network environment.

Therefore, instead of directly modifying the Internet Engineering Task Force (IETF) framework [1]-[3] to provide a more suitable policy-based network management approach for the Airborne Network, we decided to implement the IETF policy-based management framework atop existing P2P protocols. The P2P protocols are more suitable for the Airborne Network environment than the client-server architecture typically used for fixed-infrastructure networks.

We selected the JXTA peer-to-peer architecture implementation for policy-based airborne networking [5]. JXTA is a P2P development platform (i.e., an API) which provides discovery, advertisement, distribution, and message exchange services, which are appropriate to implement policy-based network management for the Airborne Network. The availability of the API gives us a head start on developing a P2P policy-based management approach.

The key features of JXTA [8] which are used in most JXTA applications are peers, peer groups, pipes, advertisements, and discovery services. Peers are the basic unit of JXTA and both produce and consume services in P2P applications. Peers self-organize into peer groups, which allow all the peers in a group to communicate. The basic tool JXTA uses for

communication within the group are pipes; they allow peers to send data messages to other peers, similar to a transport-layer UDP socket [10]. The basic JXTA pipe is asynchronous, unidirectional, and unreliable. Therefore, a JXTA peer needs both an input pipe and an output pipe to send and receive data from another JXTA peer. The pipes are not considered reliable because the JXTA protocol does not require receiving peers to acknowledge messages it has received, so messages that are lost will not be retransmitted by JXTA itself. However, developers can build acknowledgements into their applications to create the reliability required for certain applications.

In order for peers to self-organize into groups, which make communication possible, they need to advertise the services they provide. For example, a peer providing files would need to publish advertisements for both the file-sharing service peer group and its own input pipe, which it uses to receive request messages. Before being able to make a request, a peer seeking a particular file must first find and join the group that is performing the file-sharing service, and then locate an available request pipe. The peer seeking services solicits the network for available peer groups and pipes, and as a response it receives advertisements published by peers providing services. Out of these advertisements, the seeking peer selects peer groups appropriate to the desired services and chooses pipes to which to send requests. These advertisements are special XML documents that announce the presence of JXTA resources. The process of publishing, soliciting, and advertising resources is called JXTA discovery. Further details on JXTA can be found in [5].

From an Airborne Network perspective, the issues of policy format, policy storage, and policy translation are secondary to the issues of reliably distributing policies throughout the network, effectively addressing dynamic topology changes, and seamlessly allowing maximum mission flexibility. The use of P2P enables us to address these key issues first and foremost. Also the JXTA P2P protocols are flexible enough to accept any message format for policy-based management.

To implement the P2P architecture for policy-based management we considered two approaches: (1) The Low Overhead (LOH) Architecture and (2) The Discovery-Based Distributed (DBD) Architecture. The ultimate goal in implementing these two architectures was to provide a capability as shown in Figure 2. Specifically, we wanted the architectures to allow the following basic sequence of events to occur:

1. Each of the PDPs in the network discovers each other and distributes the policy repository.
2. Each of the PEPs discovers the PDPs within the network.
3. Each of the discovered PDPs responds to the PEPs.
4. Each PEP selects a PDP from which to obtain policies.
5. Each PDP tabulates or logs its associated PEPs.
6. Each PEP requests a policy configuration.
7. Each PDP sends the appropriate policies to each of its PEPs.
8. Each PDP monitors its associated PEP configurations.

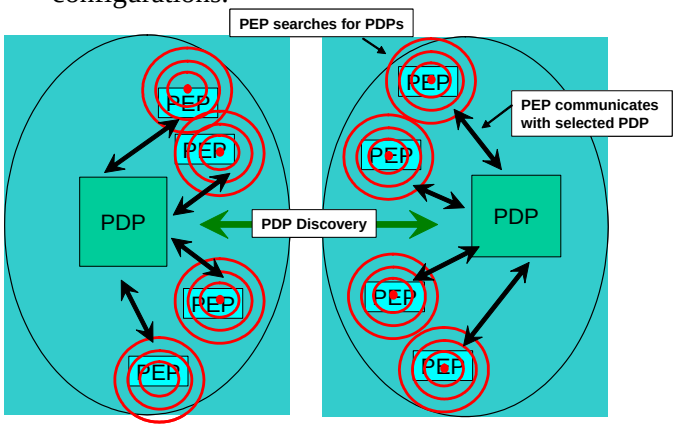

Figure 2. Message Exchange Implementation between the PDPs and PEPs.

III LOW OVERHEAD (LOH) ARCHITECTURE IMPLEMENTATION

The LOH architecture is based on the IETF framework but uses the JXTA protocols as the data transportation mechanism with minimal message exchange in order to maintain low overhead. Basically, given the low bandwidth of tactical links relative to the commercial environment, performing polling or any complex message exchanges was considered to be wasteful of the limited available bandwidth. To help maintain a low overhead, this approach used the more reliable bidirectional pipe versus the unreliable unidirectional pipe [5].

Assumptions and Limitations

In general the LOH architecture provides a very simple method to implement policy-based management via the JXTA P2P architecture. The LOH architecture uses a single peer group, which includes both PDPs and PEPs. The architecture allows only one PDP to be active at any time. Any other PDPs would act as stand-by PDPs. We assume that all PDP and PEP entities are contained within the same subnet. The policy repository is assumed to be in the same machine as the PDP. The PDP and the PEPs will use the same pre-defined pipe advertisement in XML format to establish a connection. If a stand-by PDP takes over for the originally active

PDP, the same pipe advertisement is used by the PEPs and the new PDP to establish a connection.

Clearly, the main assumption for this architecture is that all PEPs can communicate with the single active PDP in the subnetwork. This assumes that the PDP's pipe advertisement can be predefined. Predefinition of the pipe advertisements may be a reasonable assumption for the Airborne Network, where, most likely, the widebody aircraft would be the PDPs. A predefined pipe advertisement also addresses security concerns within the DoD environment, where in this case the nodes must know the "secure codes" (e.g., pipe advertisements) before two-way communication can be initiated. Since there are not many widebody aircraft, predefining the PDPs may be a reasonable operational scenario. This does limit the flexibility and the dynamic capabilities of this architecture. However, this architecture has limited overhead, since there is no polling of the peers to determine their presence in the network.

The architecture also assumes that if a PEP is in the network, it will continue to look for the active PDP until it finds it. The PDP responds to any requests for policy. Also, for purposes of this architecture the policy repository is assumed to be static. In which case, each PDP must have the same policy file, with the capability to update the policy file when necessary. Again this is a reasonable assumption, depending on the operational scenario. That is, if the operational scenario is such that a comprehensive set of policies is loaded into each PDP before a mission begins, then the PDPs simply provide the appropriate policies to the PEPs. If a policy change is needed, it is assumed that that policy change is anticipated and is already in the comprehensive set loaded in the PDPs; in which case the PDPs can "push" a new policy to the PEPs. Again these policies have to be known ahead of time. The most efficient way to perform a "push" would be to have the repository send a policy information change status message, including associated files, to the PDPs; then the PDPs would "push" the policy information to the corresponding PEPs for an instantaneous policy change.

The LOH architecture is dynamic in the sense that the PEPs can be updated with different policies from a predefined or preloaded library of policies contained in each PDP. The library does not change during a mission, but different policies from the library can be loaded into the PDPs for different mission phases. The architecture is distributed in the sense that, if we assume that different groups of PEPs (e.g., fighter groups) operate within different subnets, then each subnet of PEPs joins up with its associated PDP (e.g., a widebody) and obtains the appropriate mission policy.

The difficulty here would be how effectively policies could be changed on-the-fly. This gets into the issue of the number of subnets, the number of PDPs per subnet, operational scenarios for distributing policies, etc., which is beyond our scope at this point.

PDP Functionality

First, the PDP instantiates the JXTA platform and creates the default net peer group. Then, it uses a server pipe to accept connections from the other JXTA nodes. There is only one active PDP within a subnet. Once the PDP is up and running, it applies the JXTA discovery service. By utilizing the JXTA discovery service, a PDP searches for its peer neighbors: the PDPs or the PEPs. From the response packets of the peer discovery, a PDP becomes an active or a stand-by PDP. By reading in a pre-defined pipe advertisement, the active PDP sets-up a pipe listener within the group. The pipe advertisement lists server name, type of connection, (e.g., unicast), and the name of the server pipe etc.

PEP Functionality

Once a PEP is up, it uses the pipe advertisement (i.e., the same one that the PDP used) to set-up a connection with the PDP. Once connectivity between the PDP and the PEPs has been established, the PEP attempts to obtain the policy file from the PDP by sending a request packet. If there is any change in the policy file(s) in the repository or a change in a mission, the network management software initiates a "push" of the policy from the PDP to the PEPs.

The PEP creates a directory with the configuration file and the cached service advertisements from other peers. Once the PEP receives its policy file, it executes the script file in Linux tc format in our lab test scenarios. Linux tc is a traffic control tool which is used to configure the Linux kernel to accomplish the shaping, scheduling, policing, and dropping of packets [7].

Interaction Between the PEPs and a PDP

There are three types of messages for the PEP, the PDP and the network management console.

1) Policy Configuration Request: once communication has been established between a PDP and a PEP, the PEP will make a request to its PDP for the policy configuration file. According to the router type and the platform type, the corresponding policy file will be sent to the PEP.

2) Peer Information Request: a network management console could send out a peer information request to a PDP for all activities between the PDP and

the PEPs. Transition between the PDP and its PEPs is also logged to a file.

3) "Push" of the Policy Information: if there is a change in any policy file or a change in a mission, a network management console can connect to the PDP and request to "push" the latest network policy files from the PDP to all corresponding PEPs.

IV DISCOVERY-BASED DISTRIBUTED (DBD) ARCHITECTURE IMPLEMENTATION

Discovery-Based Distributed (DBD) Architecture

The DBD architecture addresses some of the limitations of the LOH architecture by introducing JXTA discovery mechanisms to the process of forming relationships among PDPs and PEPs. PDPs automatically discover and join a peer group, into which they publish self-generated pipe advertisements to be used for policy repository requests. Meanwhile, PEPs and PDPs together join a separate peer group, in which PDPs publish pipe advertisements to receive platform policy requests and PEPs discover them.

The DBD provides policy-based management by taking advantage of the dynamic discovery features of the JXTA protocols. Here, given the dynamic nature of the Airborne Network, we wanted to develop a policy-based management scheme that would provide a large amount of flexibility, due to the dynamic nature of the system.

Because PEPs dynamically solicit the network to discover available PDPs, multiple PDPs can be active within the same subnet. The self-generated pipe advertisement of each PDP prevents the conflicts created by the LOH architecture's single, pre-generated advertisement. This not only allows PEPs to select from a redundant field of active PDPs, but also to seamlessly transition from one PDP to another as network conditions require. The resultant architecture distributes the burden of policy distribution among all PDPs in the network.

However, the drawback of such a dynamic system is that it requires a measure of multicast soliciting and unicast polling to permit nodes to locate one another and to ensure that their own policy information is current. Also, the unidirectional pipe used by the DBD architecture is less reliable than the bidirectional pipe used by the LOH architecture. While the LOH architecture avoids the additional overhead from multicast soliciting and unicast polling, the DBD architecture's added flexibility for dynamic self-organization may make up for this shortcoming.

The high-level view of the DBD architecture is diagramed in Figure 3. A key difference from the policy-based management framework is that all PDPs have their own individual copy of the policy repository and respond to PEP policy requests with the policy information from their own repository. The PDPs are self-organized into two groups, the PDP and the PXP group. The PDP group is only used for PDP-to-PDP communication, which consists of requests for and distribution of the policy repository. The policy repository must be propagated to all PDPs whenever the policy management tool modifies it. This action requires that the policy management tool also be a member of the PDP group. The second group that the PDPs belong to is the PXP group. This group contains both PDPs and PEPs, enabling communication between them.

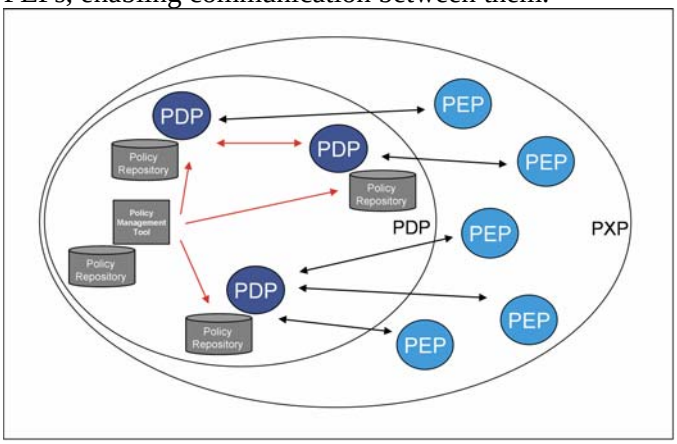

Figure 3. Overview of the DBD P2P Policy-Based Network Management Architecture.

Assumptions and Limitations

We made the following assumptions and design limitations in the development of the prototype:

- All peers are in the same subnet. This allowed us to avoid concerns about extra JXTA services, such as rendezvous and relay peers [8], which provide the necessary communications to discover JXTA peers outside of a sub-network.
- Only one policy manager would make changes to the policy repository at a given time. While the policy propagation system is distributed, control of the distributed policy database must be strictly managed. This eliminates the issue of locking repository access to prevent race conditions, if two managers make simultaneous updates.
- The PEPs are Linux routers. Due to the open-source nature of the Linux operating system, it is much easier to build and test an application on a Linux router than on a commercial router's proprietary, integrated operating system.
- The first PDP in the PDP group has a valid local copy of the policy repository. This assumption allowed us to focus development on the action of propagating the repository, rather than generating it. In a deployable

system, a policy management tool would create the PDP group, permitting the first PDPs to acquire the policy repository from it.

- PEPs select pipe advertisements randomly. It will most likely select the first pipe advertisement it receives from the PDPs. The PEP does not make any attempt to select the optimal PDP.
- Repository propagations between PDPs do not fail. Our current policy repository distribution protocol does not guarantee all PDPs have the most current policy repository and assumes that they do.
- The size of policy repository is reasonably small. Every PDP node can store a complete copy of the repository, and it can be broadcast to all other PDP nodes without severe network load.
- PEPs and PDPs do not use authentication when joining the PXP and PDP groups. JXTA provides authentication for joining groups, but to simplify the implementation we did not incorporate authentication. We assume adding it to the prototype would be fairly straight-forward.
- PDP, PEP message exchanges use unreliable unidirectional pipes. The basic JXTA pipe does not acknowledge receipt of messages. The application is responsible for providing some form of robustness.

PDP Functionality

The functions of the PDP are to discover and join the PDP and PXP groups, to acquire or initialize the policy repository, to process PDP policy repository requests and updates, and to service PEP policy record requests. The PDP has two main phases of operation, initialization and run. During the initialization, the PDP finds or creates the necessary groups and acquires the policy repository. Following that, the run phase consists of the PDP listening and responding to other PDP requests for the repository, listening for policy repository updates from other PDPs, and responding to PEP policy requests.

During initialization, the PDP dynamically discovers the policy-based network management groups, PDP and PXP. If the PDP cannot find either group it assumes that it is the first PDP, which means the PDP and PXP groups do not yet exist; thus, it creates them.

To find the necessary groups the PDP discovers peer group advertisements for the PDP and PXP groups. The discovery process is initialized with the JXTA discovery service's getRemoteAdvertisements method. The responses, if there are any, from other peers are advertisements matching the request. The discovery responses are received asynchronously after the discovery process is initialized, so the PDP waits a set amount of time (i.e., approximately 30 seconds) to see if it discovers the necessary group advertisements. If there is no response, the PDP will then create and publish the PDP and PXP groups. Figure 4 shows the interaction of the PDPs to discover the PDP group.

After joining the two groups, the PDP initializes the policy repository either from a local copy, if it is the first PDP, or by requesting the repository from another PDP.

To request the policy repository from another PDP, the new PDP needs to find another PDP who has the policy repository and then request it. Figure 5 shows the policy repository request pipe discovery and repository request process performed by the new PDP. A PDP indicates that it has a policy repository to provide by publishing a dynamically generated advertisement for its policy repository request input pipe, which it dynamically created during initialization. The PDP's policy request pipe is simply an input pipe that it creates to listen for repository requests. Similar to the process for discovering peer group advertisements, the new PDP, requesting the policy repository, uses the JXTA discovery service to find policy request input pipe advertisements. Upon receipt of the pipe advertisements, the new PDP selects one from the collection of responses. Using the newly discovered input pipe the new PDP dynamically creates an output pipe to send a request message to the PDP with the repository.

However, before the new PDP can send the policy repository request message it needs to have a means to receive the response (i.e., the policy repository). All PDPs receive policy repositories through the policy distribution pipe. This pipe is a JXTA propagate pipe whose static advertisement is preloaded as a file on all PDPs. The propagate pipe is one of two modes of communication provided by JXTA pipes. The first mode is the standard point-to-point pipe, which connects one input endpoint to one output endpoint. On the other hand, the propagate pipe connects one output endpoint to multiple input endpoints, effectively creating a method to broadcast messages to all listening peers. Using the propagate pipe provides a simple means for all PDPs to receive policy repository updates with a single update message.

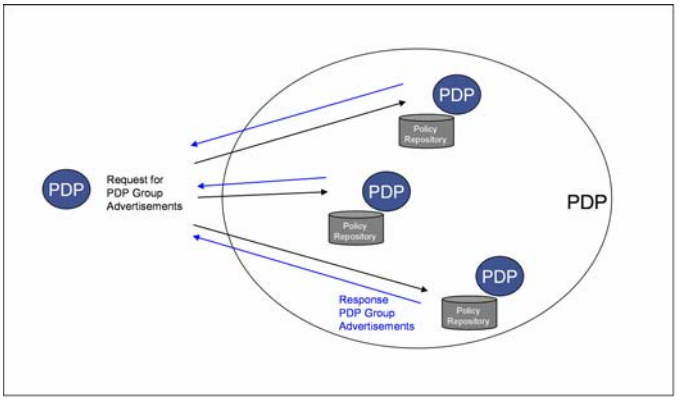

Figure 4. PDP Group Discovery Process for the DBD Architecture.

Even though the new PDP requested the policy repository, all PDP peers will receive the policy repository, because they all created input endpoints to the policy repository distribution pipe. This adds some robustness because pipes are unreliable and there is no way for a PDP to determine or guarantee that its repository is the current version. When a PDP receives a policy repository due to a request from another PDP, it can check the received version against the version of its current repository. If its current repository is outdated then it will update its repository with the received version. If the received policy repository happens to be older then the current one then the PDP will discard it. Such a condition indicates the existence of a PDP that failed to receive previous policy repository propagations, and by chance it received a policy repository request and distributed its outdated copy. Currently, the only mechanism that can repair the PDP with an old policy repository is to wait for it to receive future repository distributions. We realize that this implementation does not have acceptable functionality for a deployable policy-based network manager; but for the proof-of-concept prototype, it simply demonstrates the ability to distribute the policy repository. Further research is needed to develop a more robust repository distribution protocol that can account for and correct all out-of-date PDPs.

Now that the new PDP has set-up an input pipe to receive the policy repository, it sends the request for the repository through the policy request input pipe that it discovered. Once the request message is received, the policy repository is sent out through the policy repository distribution pipe, and all PDPs including the initializing PDP will receive the policy repository. Now that the new PDP has a policy repository, it can begin listening for policy repository requests. The PDP does this by creating and publishing its policy repository request pipe. At this point, the new PDP has its own copy of the policy repository and is capable of receiving requests for the repository from other PDPs. Figure 6, provides a graphical representation of the PDP-to-PDP communication pipes.

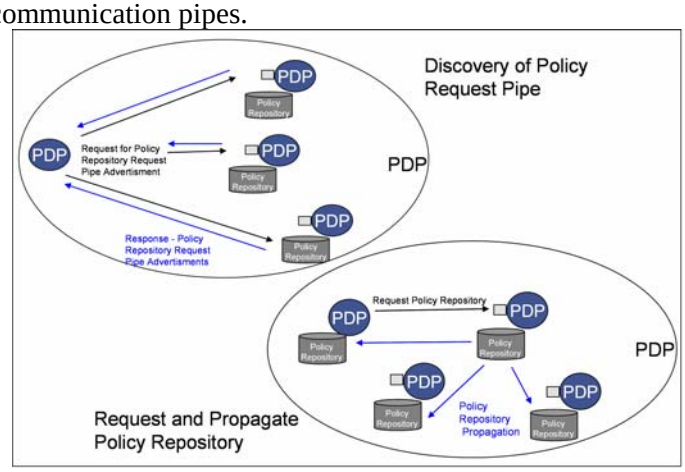

Figure 5. Policy Repository Request Process for the DBD Architecture.

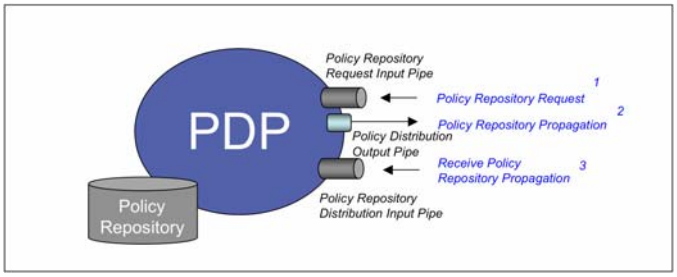

Figure 6. Detail of PDP-to-PDP Communications for the DBD Architecture.

Once the PDP has acquired a policy repository, it now has everything required to manage policies for PEPs. However, before initialization is complete, the PDP must create and publish a policy request input pipe advertisement, in order to receive policy request messages from the PEPs. After constructing the policy request input pipe, the PDP enters the main phase of operation.

The run phase is an execution loop that listens for PDP policy repository requests and PEP policy requests on the respective pipes. In the initialization phase, the PDP first published its pipe advertisements informing other PDPs and PEPs of its services. Because of the expected dynamic nature of the network it is best to give these advertisements a short lifetime to limit the number of stale advertisements in the P2P network. So, at set intervals during the run phase, the PDP freshly republishes its pipe advertisements to the groups. This ensures that other PDPs and PEPs know the PDP is still present on the network and is listening for requests. Also in the run phase, the main task of the PDP is to respond to the requests it receives from other PDPs and PEPs.

Policy Repository Data Structure

The prototype system structures the policy repository as a list of platform policies. This list is implemented as

a Java HashMap, which uses the platform type and mission phase as the index to each platform policy. In order to determine if a received policy repository is newer than what the PDP already has, the policy repository includes a version field which is merely a timestamp.

The platform policy is an object that contains multiple strings to hold the values for platform policy name, version, platform type, platform ID, mission phase, and router type. The different pieces of information contained in the platform policy help to identify the appropriate policy. Therefore, each individual aircraft platform will have a policy for each possible phase of the mission, and possibly for each type of router.

In addition to the platform information, the platform policy also contains two policy objects: one for the In policy and another for the Out policy. The In policy applies the configuration contained in the policy object to the incoming traffic and, conversely, the Out policy applies the configuration to the outgoing traffic. The configuration information contained in the policy object is simply two text strings for both the filter command and the action command. The Linux 'tc' and 'ipfilter' tools would be used to generate the appropriate filter and action commands to implement the policy. Figure 7, is a Unified Modeling Language (UML) diagram for the policy repository data structure.

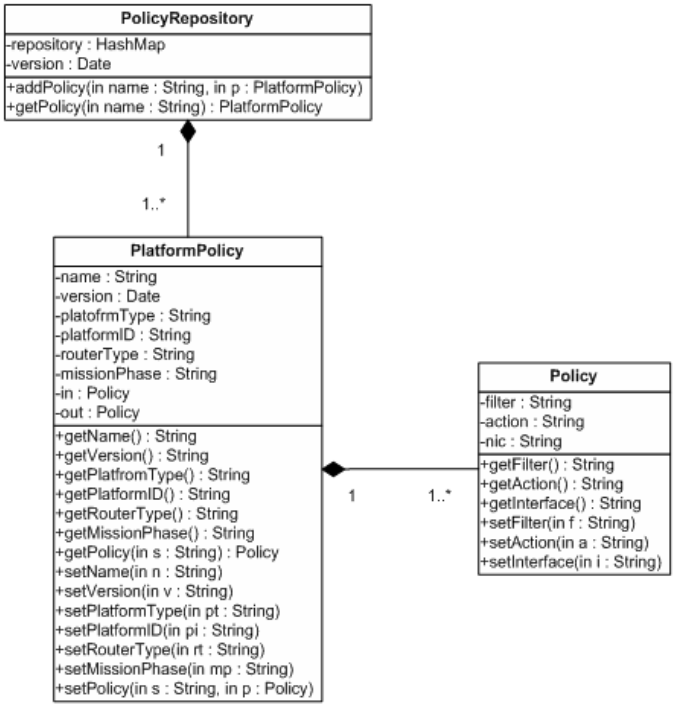

Figure 7. Policy Repository Data Structure for the DBD Architecture.

PEP Functionality

The functions of the PEP are to discover and join the PXP group, and to discover a PDP. After a PDP has been found, the PEP sends policy request messages to the PDP on a configurable interval. If the PEP fails to send a request message to the PDP, the PEP returns to PDP discovery and waits until it has found another PDP. Once one is found the PEP resumes sending policy request messages.

Similar to the PDP, the PEP also has an initialization phase and a run phase. The main purpose of the initialization phase is to find and join the PXP group as well as create its policy input message pipe. If the PXP group cannot be found, the PEP remains in the initialization phase until it locates the group. Once the PEP has joined the group, the final initialization step is for the PEP to create its policy input message pipe. The PEP differs from the PDP in that the PEP's policy input message pipe does not need to be published, because the PEP will send the pipe advertisement directly to the PDP. The direct messaging of the advertisement means this pipe will not be discovered; therefore the pipe advertisement does not need to be published.

Initialization is completed after the PEP has created its policy input message pipe. After initialization, the PEP enters the main mode of operation, which consists of an execution loop with two states. The first state searches for PDP policy input message pipe advertisements, and uses the discovered advertisement to create an output endpoint to the PDP's policy input message pipe. The second state uses the output pipe to periodically send policy request messages to the PDP and check the response to those messages. If the PEP fails to receive a response from the PDP after time has elapsed for sending another policy request, the connection to the PDP is gone. When the connection to the PDP is lost the PEP will return to the first state of the run phase to find another PDP link.

PXP Group Discovery

The PXP group contains both PDPs and PEPs, providing a means by which PDPs can advertise platform policy request pipes and PEPs can locate them. Similar to the PDP, the PEP uses the JXTA group discovery process, illustrated in Figure 8, to join the PXP group. However, if no pre-existing PXP group can be found, the PEP group must simply continue to seek it, as only a PDP can create the PXP group.

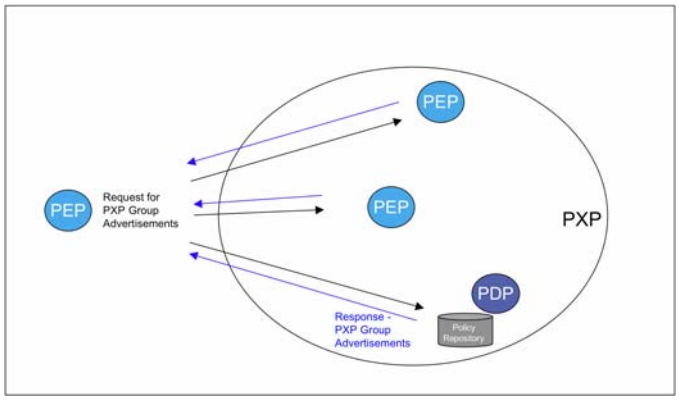

Figure 8. PXP Group Discovery for the DBD Architecture.

Policy Request and Response Process

As mentioned previously, the PDP indicates that it has a policy repository available for distribution with a policy repository request pipe advertisement; similarly, the PDP advertises the ability to manage policy for PEPs with a policy input message pipe advertisement. The PEP searches for PDP advertisements and selects a PDP that can provide policies for its configuration management.

Now that the PEP has the pipe advertisement from the chosen PDP, it can create an output endpoint to the PDP's pipe and send it policy request messages at a set interval. The PEP continues to send policy request messages to the selected PDP until it fails to receive a response. If the PDP does not respond to a policy request by the end of the interval, the PEP infers that it has lost connectivity to that PDP. Therefore, the PEP returns to the policy input message pipe discovery state and returns to the process of locating PDPs. The PEP selects a new PDP and begins requesting policies from it instead. Figure 9 illustrates the pipe discovery state and the subsequent policy request and response state.

When the PEP has dynamically discovered the PDP policy input message pipe it enters a loop where it sends a policy request message to the PDP and then waits a configurable period before sending the next policy request. The policy request message sent by the PEP basically tells the PDP the PEP's platform information, such as type and mission phase, along with the current version of its platform policy. The PDP uses the information received in the policy request message to look up the appropriate platform policy in the policy repository and determines if the PEP's version is current. If it is current the PDP simply replies with an acknowledgement message to the PEP. Otherwise, the PEP's platform policy needs to be updated with the version the PDP has in its repository.

To update the PEP, the PDP sends it an update message, which includes the platform policy as a serialized Java object. It is important to note that the PDP is only able to respond to the PEP, because the PEP included its own policy input message pipe advertisement inside the policy request message it sent to the PDP. Thus, the PDP can use the pipe advertisement to create an output endpoint to the PEP's policy input message pipe and the PDP can then send the appropriate response. The preceding PEP-PDP policy request and response process is detailed in Figure 10.

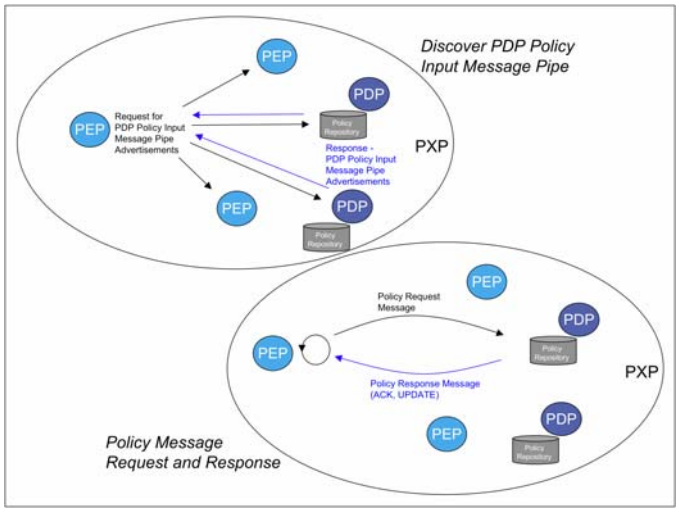

Figure 9. PEP-PDP Policy Message Process for the DBD Architecture.

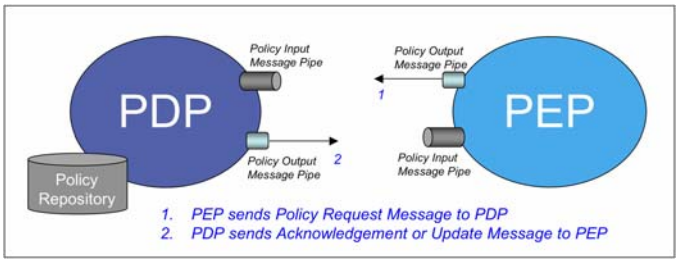

Figure 10. PEP-PDP Detailed View of Policy Request and Response for the DBD Architecture.

The PEP procedure of requesting policy updates from the PDP at a set interval provides a mechanism for the PEP to monitor its connection to a PDP. The monitoring executed by the PEP is essentially the same as a router sending out periodic "hello" messages. Since, the Airborne Network is expected to be a dynamic environment, and it is assumed that platform network connections will be intermittent at times due to platform mobility, it is important that the PEPs dynamically locate PDPs. To maintain the link, the PEPs will also need to monitor their connection with the PDP. In doing so, the PEP will be able to receive policy updates in a timely manner.

There are two possibilities for a PEP to receive a policy update. First, the PEP may be simply joining a network; or it may be changing mission phase; in either case it will make a configuration request based on its

current state. This will be referred to as the PEP "pulling" the configuration from the PDP. The second possibility is that the policy was updated by the policy management tool and the updates need to be deployed or "pushed" to all the PEPs in the network. The current PEP-PDP message procedure can accomplish both pulling and pushing of policies even though it may not be obvious at first glace. Clearly, the PEP will be pulling the policy every time it sends a policy request message to the PDP. It, of course, will only receive a policy if the PDP has a newer version. Given that the PEP is effectively attempting to pull a new policy every message cycle, any policy repository updates that are propagated to all the PDPs will be pushed to the PEP in response to its continuous pulling. This mechanism in fact creates a system that can monitor PDP connections, pull policies from PDPs, and push policies to PEPs.

Observations

Some of the initial issues discovered while developing the DBD system include the potential for PDPs to possess out-of-date policy repositories, the inability of PEPs to select optimal PDPs, our uncertainty about the quantity of overhead generated by the JXTA discovery service, and the use of unacknowledged messaging by way of JXTA's basic pipe. All of these concerns have potential solutions, which would require further analysis to determine their effectiveness.

Along with these issues it is also unknown how large a policy repository will be for the Airborne Network. If the Airborne Network requires a very large policy repository, then we would need to look at distributing portions of the policy repository to PDPs and ensuring that the separate pieces of the repository have redundant copies among the PDPs.

V CONCLUSIONS

Both the LOH and DBD architectures represent the first steps towards developing a P2P-oriented policy-based management system for the Airborne Network. Currently, both architectures operate only on a single subnet. While more development would be required to expand their capabilities, both systems illustrate that the basic capabilities of JXTA are well-matched towards providing a P2P architecture for policy-based management in a dynamic airborne networking environment. Again various capabilities can be added to both architectures to take full advantage of the JXTA P2P features, such as the rendezvous and monitoring capabilities. For example, by applying rendezvous service, PDPs would be able to communicate with other PDPs across subnets. Also, with the use of the JXTA monitoring API, we could observe the performance of network devices. These features would greatly enhance our policy-based network management capability. Also, the performance of these architectures over simulated wireless links needs to be investigated.

VI ACKNOWLEDGEMENT

The authors would like to thank Edward Palo, Christopher Nissen, Randall Landry, and Kevin Grace of the MITRE Corporation for encouraging us in this effort.

VII REFERENCES

[1] D.C. Verma, Policy-Based Networking: Architecture and Algorithms, New Riders Publishing, Indianapolis, IN , 2001.

[2] D. Kosiur, Understanding Policy-Based Networking, Wiley, New York, 2001.

[3] R. Yavatkar, D. Pendarakis, and R. Gerin, "A Framework for Policy-based Admission Control", RFC 2753, January 2000.

[4] L. Zambenedetti, M.J.B. Almeida, and L.M.R. Tarouco, "Managing Computer Networks Using Peer-to-Peer Technologies," IEEE Communications Magazine, October 2005.

[5] www.jxta.org

[6] B. Wilson, "JXTA", First Edition, New Riders Publishing, Indianapolis, IN, June 2002.

[7] http://www.knowplace.org/shaper/examples.html

[8] http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf

[9] http://www.jxta.org/Tutorials.html

[10] S. Oaks, B. Traversat and L. Gong, "JXTA in a Nutshell", O'Reilly, Sebastopol, CA, 2002.

[11] R. Schollmeier, "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications", First International Conference on Peer-to-Peer Computing, August 2001.