Distributed Policy Learning for the Cognitive Network Management System

Nathan VanderHorn and Benjamin Haan
Rockwell Collins
{navander, bjhaan}@rockwellcollins.com

Marco Carvalho, and Carlos Perez
Institute for Human and Machine Cognition
{mcarvalho, cperez}@ihmc.us

Abstract—The Cognitive Network Management System (CNMS) is a joint research initiative between Rockwell Collins and the Institute for Human Machine Cognition that aims to provide automated, policy-based real time network management for complex MANET networks. CNMS is a lightweight and efficient policy management framework designed to mitigate the need for centralized network management, reduce operator hands-on time, and increase network reliability. CNMS provides the necessary reasoning and enforcement mechanisms for the on-demand management of network topology and communication resources. Furthermore, it supports fully distributed policy learning mechanisms that enable networked devices to adapt at run-time to unanticipated network conditions and application requirements by creating and distributing learned policies.

In this paper we describe the CNMS architecture and functionality. We focus our discussions on the CNMS architecture and policy learning mechanisms. Policy learning in CNMS is intrinsically distributed, and based on network performance observations for the refinement of contexts, and actions. In this paper we describe two examples for policy adaptation, one based on link capacity monitoring, and one based on adaptive frequency hopping strategies for interference mitigation. We then present some results from NS-3 evaluations of these two examples.

CNMS has been implemented and tested in a two channel wireless testbed using the Universal Software Radio Peripheral (USRP) [1] and 802.11 wireless networking communication. We briefly discuss the testbed and describe some of our experimental results followed by a brief discussion of our findings, and recommendations for future work.

I. INTRODUCTION

Due to the complex and dynamic nature of mobile ad-hoc networks (MANETs), conventional network management approaches are generally difficult to implement and impractical to operate and maintain. The lack of a centralized management infrastructure and the dynamic nature of the network makes it unrealistic to use centralized services for policy reasoning and dissemination. Thus, policy services for tactical network environments must be distributed, agile and adaptive to different operational conditions and resource availability. Inspired by this need, and by previous research in Policy-Based Network Management [2] [3] and Cognitive Network[4], we have developed a modular and extensible cognitive network management system (CNMS) framework.

The primary goal of CNMS is to move the burden of network management away from the network operator and into the networked device. That is, nodes in a cognitive network have the ability to monitor local network metrics and automatically adjust their operational parameters to enhance individual and network wide system performance, not only following operational and regulatory policies, but also learning, from experience, new operational policies that can be used in the future by other nodes in a similar context.

In a cognitive network each node in the system is responsible for monitoring local network behavior and adjusting operational parameters based upon mission policies. In addition, cognitive nodes have the ability to learn new policies that can be shared with other nodes, improving their ability to adapt to similar network conditions in the future. To achieve this goal, CNMS-enabled nodes build an experiential knowledgebase from node-state information and operational context. State information is locally provided by a node’s Engines, which are independent components providing data such as GPS position, spectrum availability, terrain information, quality of service (QoS), and other network statistics. Each node also has a mission description that defines its objective tasks and optimization criteria, as well as a set of policies that define its regulatory constraints and operational parameter bounds. Armed with that information, each node seeks to optimally allocate resources to enable or enhance overall network qualities such as topology connectivity, reliability, robustness, etc.

Currently, most solutions applied to network planning and policy management are done offline, as part of a pre-planning phase, and tend to rely on background knowledge and expertise from network managers and operators. The goal of the CNMS program is to create a management system that mimics the decisions typically made by network operators. This will reduce the operator hands on time and create a much more robust network.

II. CNMS NETWORK ARCHITECTURE/PARADIGM

Our approach to MANET networking considers a scenario where nodes have multiple interfaces, possibly connected to highly heterogeneous links or tactical radios. Furthermore, the proposed system is capable of using multiple waveforms, each having different characteristics. That is, we propose that no single waveform will accommodate all network traffic without (potentially significant) compromises, particularly in resource constrained environments.

The concept behind our Network Architecture is to augment a homogeneous mesh network with additional waveforms that can be used to fulfill specific data transport requirements. That is, instead of developing a single waveform capable of
handling all network traffic, we utilize multiple waveforms optimized for specific purposes (latency, bandwidth efficiency, range, etc.). The intent is for the mesh waveform to be used to exchange pertinent network metrics and provide only limited user data payload support. The heavy lifting is then accomplished via on demand support waveforms. In this document we will collectively refer to these additional waveforms as the “socket” waveform(s). For our existing testbed, as discussed below, we have chosen 802.11 as our mesh communication network and utilize the USRP as the reconfigurable, on demand socket waveform.

III. CNMS ARCHITECTURE

CNMS is subdivided into 3 main layers, as illustrated in Figure 1. The first layer consists of a series of engines that supply tools and input to the system. Engines come in two flavors, sensor engines and learning engines. Sensor engines collect, filter, and report specific information to the node’s knowledgebase. An example of a sensor engine is the position awareness engine. This engine will collect GPS information from the radio as well as surrounding radios in the network and can influence waveform selection based upon range estimates. Another example of a sensor engine is the spectrum engine. This engine is attached to an XG sensor and will collect information about desired portions of the available spectrum. Learning engines supply tools that the system can use to do more advanced tasks such as optimization and clustering. An example of a learning engine is the optimization engine. This engine is a tool that can be used to optimize system settings such as transmitter power, detection thresholds, and transmission rates using learning techniques such as genetic algorithms [5], [6], and particle swarm optimization [7].

The second layer of CNMS is again subdivided in three parts, an XML knowledge base [8], a policy reasoner, and a mission management component. The XML knowledge base stores system information and policies in an XML format for easy analysis and transmission to other CNMS nodes. The goal of the policy reasoner is to make quick decisions based on the current system state and a set of predefined and/or learned policies. CNMS policies cover a large range of managerial tasks and represent many concepts. The mission management component controls the operation of the overall system based on a set of mission files. By changing missions the operation of the system can be modified. Missions control what policies are loaded and what engines are running at any given time. Missions also control the flow of data throughout the system and control how often tasks such as optimization are performed.

Finally, information and actions from the policy/mission layer get fed into the third layer (cognitive layer) of the CNMS system. The goal of the cognitive layer is to analyze the effects of decisions made by the myopic lower layers and modify policies to improve long term system performance. The following sections describe some of the functionality of the cognitive layer in more detail.

IV. POLICY LEARNING

The CNMS can operate as a strict policy based management system, however, using learning techniques, the system is able to automatically generate new policies and/or adapt existing policies to improve system performance. The following two sections describe two scenarios in which we used learning techniques to enhance network performance.

A. Mesh-Network Capacity Monitoring

One of the key network management decisions when using our network paradigm is to determine when to transfer user data over the mesh and when to open a socket channel. As mentioned, the main priority of the mesh is to pass control and status information throughout the network. However, under low traffic conditions, it is possible to transfer payload data through the mesh network without incurring the penalty of opening a socket channel. At some point, this payload data may burden the mesh and must be transitioned to a socket. In this case, a policy is created to signal socket creation when traffic on the mesh exceeds a certain threshold. The firing of the policy depends on the threshold for the capacity of the mesh network. This threshold can vary depending upon the utilization of the mesh.

An effective policy adaptation mechanism in this scenario is to find the best threshold for mesh network capacity for the current network conditions. For that purpose we identify (or establish) costs for sending data above the capacity of the mesh network (i.e. dropping packets), as well as costs for establishing the socket.

Our solution for learning the threshold is to use a reinforcement learning approach known as Q-Learning [9, p.148]. The problem of finding the threshold can be specified as a reinforcement learning problem in the following way:

1) Actions:
   - Mesh: Use the mesh link
   - Socket: Use the socket link

2) State: Transmission data rate in Mbps (discretized to intervals of 1Mbps).
3) **Cost**: The percentage of packets dropped for the respective interface (mesh or socket).

As the socket link is a dedicated channel with a much higher capacity, if we only use packet drop percentage to determine the cost, then the trivial solution would be to always use the socket link. For that reason we add a constant cost to the socket link.

The proposed approach was tested with simulations for 2, 6 and 10 nodes. In each scenario, half of the nodes are senders and the other half are receivers. All nodes share the aggregate capacity of the mesh network and compete for its bandwidth. In our simulations, every 10 seconds the sending nodes uniformly select a TX data rate between 1Mbps and 10Mbps. The theoretical capacity of the mesh link is 6Mbps. The expected outcome is that, as more nodes are sending data on the mesh network, the value for the mesh link on the Q-Learning function will degrade, making us reduce the threshold for creating the socket link.

Each scenario was run 5 times, and the Q-Learning value function for each node on each run of each scenario was extracted and plotted in figures 2, 3, and 4.

As expected, with a higher number of nodes, the value function for the mesh link decreases, reducing the threshold for creating the socket. If, for example, the cost for the socket link is set to 20%, then for 2 node scenario the threshold is 6Mbps, because for all rates below 6Mbps, the value function is lower than 0.2. Similarly for 6 nodes, the threshold is 3Mbps; and for 10 nodes, it is 2Mbps.

**B. Socket Robustness to Localized Interference**

Another case in which we demonstrated the benefits of CNMS was after a socket has been established. Because the socket waveform is configurable, we wanted to understand if/how we would mitigate RF interference.

Consider that, upon detection of RF interference, a decision must be made on whether to stay or jump to a different channel (previous studies have shown that a reactive strategy for jamming avoidance is better than a proactive strategy [10]). That is, if we detect a sweeping jammer, we should remain on the channel and wait for the jamming signal to pass. If, on the other hand, the jammer is single channel interference the best action is to jump to a new frequency. An effective policy learning strategy for this scenario will identify a strategy that would minimize the effects of interference/jamming, for a given interference profile.

In this scenario the CNMS agent periodically checks the
packet drop rate for each socket link. Let’s call each time the algorithm runs a timestep. Whenever the drop rate on a timestep reaches 50%, it is considered that the link has been hit by a jammer. When the CNMS detects a hit, it can take 2 possible actions, stay or jump. The idea is to determine, for an unknown jamming strategy what’s the best action to take.

To measure the effectiveness of the learning, we define a metric that indicates how well the algorithm avoids being hit by the jammer. This metric is the proportion of timesteps that a hit to the link is detected (when packet loss ≥ 50%). The lower the metric, the more effective the learning. For the baseline metric we first determine how well an agent that always follows the baseline metric we first determine how well an agent that always follows the baseline strategy, will get hit by a jammer. This metric is the proportion of timesteps that a hit to the link is detected (when packet loss ≥ 50%). The lower the metric, the more effective the learning.

For measuring the effectiveness of the proposed learning algorithm we ran simulations where a jammer was randomly changing its jamming interval every 10 jumps. This simulation was run 10 times, and not only for the learning agent, but also for the agents with the fixed strategies (stay agent or jump agent). Table II shows the metric values for the simulations (the jammer selected a jamming interval between 1s and 10s), each row represents an execution of the simulation.

Figure 5 shows the plot for this scenario. In this scenario the adaptive agent and the jump agent obtained better results than the stay agent, but the adaptive agent has a much lower deviation. For this scenario the expected value for the jammer interval is 5s, making the jump strategy better on average.

Now, we consider a scenario with a smaller jamming interval. Table III shows the metric values for the simulations with a jammer selecting a jamming interval between 1s and 8s. In this case the expected jamming interval is 4s, making, in theory, both strategies equally effective.

Figure 6 shows the plot for this scenario. In this scenario the adaptive agent is better than both fixed strategy agents. But again the jump agent obtained better results than the stay agent, and in theory they should have had obtained similar results. The reason for this is that as the metric is based on the number of hits, and a hit is measured in each timestep, with higher jamming intervals, the jump agent will get a chance to run longer (remember that the jammer changes intervals every 10 jumps).

<table>
<thead>
<tr>
<th>Jamming Interval (s)</th>
<th>Stay</th>
<th>Jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>1s</td>
<td>0.249294 0.994995</td>
<td></td>
</tr>
<tr>
<td>2s</td>
<td>0.250250 0.497497 0.250250 0.199199</td>
<td></td>
</tr>
<tr>
<td>3s</td>
<td>0.250250 0.332323 0.250250 0.199199</td>
<td></td>
</tr>
<tr>
<td>4s</td>
<td>0.250250 0.249249 0.250250 0.199199</td>
<td></td>
</tr>
<tr>
<td>5s</td>
<td>0.250250 0.199199 0.250250 0.199199</td>
<td></td>
</tr>
<tr>
<td>6s</td>
<td>0.252252 0.166166 0.252252 0.142142</td>
<td></td>
</tr>
<tr>
<td>7s</td>
<td>0.252252 0.142142 0.252252 0.125125</td>
<td></td>
</tr>
<tr>
<td>8s</td>
<td>0.252252 0.125125 0.252252 0.111111</td>
<td></td>
</tr>
<tr>
<td>9s</td>
<td>0.252252 0.111111 0.252252 0.100000</td>
<td></td>
</tr>
<tr>
<td>10s</td>
<td>0.252252 0.100000 0.252252 0.090909</td>
<td></td>
</tr>
</tbody>
</table>

Table I shows some results for a jammer with several fixed hopping intervals (each row is a different jamming interval and a different simulation). The timestep for the CNMS agent is 1s (the jammer selected a jamming interval between 1s and 10s), each row represents an execution of the simulation.

Figure 5. Plot for data from Table II
Finally we consider a scenario with an even smaller jamming interval. Table IV shows the metric values for the simulations with a jamming interval between 1s and 6s. In this case the expected jamming interval is 3s, making in theory the *stay* strategy more effective. Figure 7 shows the plot for this scenario. In this scenario the *adaptive* agent is better than both fixed strategy agents. But this time, as predicted, the *stay* agent obtained better results than the *jump* agent.

### V. Experimental Testbed

A picture of the CNMS testbed is shown in Figure 8. The testbed consists of 5 cognitive nodes. Each node has a standard 802.11 wireless card and a USRP. As mentioned, all nodes are connected in a mesh configuration using the 802.11 link and on demand sockets can be opened using the USRP. In addition, all nodes run a version of the CNMS software. Our testbed demonstration consisted of 4 experiments to illustrate the flexibility and use of CNMS. The following sections describe the experiments in more detail.

#### A. Dynamic socket creation

The first experiment is to demonstrate mesh network traffic monitoring and dynamic socket creation as discussed in section IV. The system is initiated with very low load traversing the 802.11 mesh, only CNMS generated traffic such as GPS position exchange, socket availability and network usage metrics are consuming mesh bandwidth. Traffic is then added to the mesh in the form of a streaming video. As a predefined policy, we force any node to open a socket when the traffic sourced to the mesh is over 500kbps. Thus, when the 1Mbps streaming video is launched, initially over the mesh, the policy is fired and as a result, a socket is immediately opened and the traffic is redirected over the socket link. We note here that we could use the Q-Learning method discussed earlier to adapt the 500k threshold value based upon real mesh loading, however, this feature has yet to be implemented into the testbed.

#### B. Terrain Awareness

As part of the CNMS architecture, a terrain awareness engine is built into the system. The terrain awareness system uses digital terrain elevation (DTED) data and corresponding node position to assist in line of sight (LOS) calculation between CNMS nodes. For demonstration purposes, the testbed uses artificial GPS information and node position is displayed to a World Wind [11] geographical map to visualize movement. A graphical depiction of the scenario is shown in Figure 9. To demonstrate terrain awareness, Node 1 first opens a socket to Node 2. As the nodes continue on a path around opposite sides of a building, the terrain changes, forcing the nodes to adapt their strategy to maintain connectivity.

---

**Table III**

<table>
<thead>
<tr>
<th>Adaptive</th>
<th>Stay</th>
<th>Jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.184184</td>
<td>0.249249</td>
<td>0.202202</td>
</tr>
<tr>
<td>0.185185</td>
<td>0.232232</td>
<td>0.215215</td>
</tr>
<tr>
<td>0.193193</td>
<td>0.262262</td>
<td>0.202202</td>
</tr>
<tr>
<td>0.193193</td>
<td>0.251251</td>
<td>0.234234</td>
</tr>
<tr>
<td>0.204204</td>
<td>0.238238</td>
<td>0.256256</td>
</tr>
<tr>
<td>0.173173</td>
<td>0.229229</td>
<td>0.213213</td>
</tr>
<tr>
<td>0.205205</td>
<td>0.237237</td>
<td>0.208208</td>
</tr>
<tr>
<td>0.199199</td>
<td>0.236236</td>
<td>0.213213</td>
</tr>
<tr>
<td>0.200200</td>
<td>0.237237</td>
<td>0.212212</td>
</tr>
</tbody>
</table>

**Table IV**

<table>
<thead>
<tr>
<th>Adaptive</th>
<th>Stay</th>
<th>Jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.218218</td>
<td>0.242242</td>
<td>0.251251</td>
</tr>
<tr>
<td>0.222222</td>
<td>0.233233</td>
<td>0.316316</td>
</tr>
<tr>
<td>0.221221</td>
<td>0.232232</td>
<td>0.258258</td>
</tr>
<tr>
<td>0.228228</td>
<td>0.235235</td>
<td>0.323233</td>
</tr>
<tr>
<td>0.233233</td>
<td>0.224224</td>
<td>0.286286</td>
</tr>
<tr>
<td>0.230230</td>
<td>0.232232</td>
<td>0.310310</td>
</tr>
<tr>
<td>0.226226</td>
<td>0.224224</td>
<td>0.294294</td>
</tr>
<tr>
<td>0.226226</td>
<td>0.228228</td>
<td>0.315315</td>
</tr>
<tr>
<td>0.223223</td>
<td>0.239239</td>
<td>0.292292</td>
</tr>
</tbody>
</table>
of the mountain CNMS constantly checks the intervisibility between the nodes. When LOS becomes occluded, the terrain policy is fired and notifies the nodes to search for an alternate path. As a result, a multihop socket is created via Node 3 which is loitering over the top of the mountain. LOS is continually checked between nodes 1 and 2 and, when they once again are within LOS, the single hop socket is restored.

C. Automatic socket type adaptation

The third experiment demonstrated with the CNMS testbed is the automatic socket adaptation. As mentioned before, the socket link is actually a conglomeration of many different waveforms - each having different performance characteristics. To demonstrate the flexibility of CNMS we have implemented socket type adaptation. For this experiment we have encoded areas of the terrain map as foliage covered terrain. We begin with a Ku frequency band socket between nodes 4 and 5. Although Ku band waveforms exhibit very high bandwidth characteristics due to the abundance of spectrum, they are not well suited for operation in foliage covered terrain. Thus, in our demonstration scenario, node 5 moves to a position which has been coded as high foliage. During socket operation, the CNMS terrain policy constantly monitors the terrain type and fires when node 5 moves into the foliage covered area. For our demo, the policy states that the socket type should be adapted from the Ku waveform to a UHF waveform. This concept can be extended to cover various terrain types and waveforms by generating policies according to specific terrain and waveform characteristics. In addition, our plan is to use machine learning techniques to learn which waveform is best suited for the existing environmental conditions.

D. Spectrum Awareness

Finally, we demonstrated the cognitive spectrum awareness features discussed in IV-B. For this experiment we simulated a jammer by injecting packet loss on the socket communication link. We simulated both a sweeping jammer and a single channel jammer. The jamming avoidance learning algorithm discussed in section IV-B was implemented and tested in the experimental testbed with results similar to the simulation results.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have introduced a light-weight Cognitive Network Management Services (CNMS) framework for tactical networks. Building from previous research in Cognitive Networks and Policy-Based Network Management, we have identified the need for a simple modular framework that can be extended to accommodate different policy engines and enforcement mechanisms, as well as the need for distributed policy learning and localized adaptation. CNMS was developed to address these needs and, while still a proof-of-concept, it has been demonstrated in laboratory settings using various policy services and different learning strategies.

As part of our continuing research, we are designing a small-footprint version of the policy enforcement and policy dissemination mechanisms for a flight demonstration and evaluation of CNMS.

REFERENCES