1-1-2011

PUMA: Policy-Based Unified Multi-radio Architecture for Agile Mesh Networking

Changbin Liu
University of Pennsylvania

Ricardo Correa
University of Pennsylvania

Harjot Gill
University of Pennsylvania

Tanveer Gill
University of Pennsylvania

Xiaozhou Li
University of Pennsylvania

See next page for additional authors

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-11-12.

PUMA: Policy-Based Unified Multi-radio Architecture for Agile Mesh Networking

Abstract

This paper presents the design and implementation of *PUMA*, a declarative constraint-solving platform for policy-based routing and channel selection in multi-radio wireless mesh networks. In PUMA, users formulate channel selection policies as optimization goals and constraints that are concisely declared using the *PawLog* declarative language. To efficiently execute *PawLog* programs in a distributed setting, PUMA integrates a high performance constraint solver with a declarative networking engine. We demonstrate the capabilities of PUMA in defining distributed protocols that cross-optimize across channel selection and routing. We have developed a prototype of the PUMA system that we extensively evaluated in simulations and on the ORBIT testbed. Our experimental results demonstrate that PUMA can flexibly and efficiently implement a variety of centralized and distributed channel selection protocols that result in significantly higher throughput compared to single channel and identical channel assignment solutions.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-11-12.

Author(s)

Changbin Liu, Ricardo Correa, Harjot Gill, Tanveer Gill, Xiaozhou Li, Shivkumar Muthukumar, Taher Saeed, Boon Thau Loo, and Prithwish Basu

PUMA: Policy-based Unified Multi-radio Architecture for Agile Mesh Networking

Changbin Liu* Ricardo Correa* Harjot Gill* Tanveer Gill* Xiaozhou Li*
Shivkumar Muthukumar* Taher Saeed* Boon Thau Loo* Prithwish Basu†
*University of Pennsylvania †Raytheon BBN Technologies

ABSTRACT

This paper presents the design and implementation of *PUMA*, a declarative constraint-solving platform for policy-based routing and channel selection in multi-radio wireless mesh networks. In PUMA, users formulate channel selection policies as optimization goals and constraints that are concisely declared using the *PawLog* declarative language. To efficiently execute *PawLog* programs in a distributed setting, PUMA integrates a high performance constraint solver with a declarative networking engine. We demonstrate the capabilities of PUMA in defining distributed protocols that cross-optimize across channel selection and routing. We have developed a prototype of the PUMA system that we extensively evaluated in simulations and on the ORBIT testbed. Our experimental results demonstrate that PUMA can flexibly and efficiently implement a variety of centralized and distributed channel selection protocols that result in significantly higher throughput compared to single channel and identical channel assignment solutions.

1. INTRODUCTION

Recently, the following trends have emerged in wireless networking: (1) transceivers supporting multiple tunable RF channels are becoming common; (2) devices with multiple wireless interfaces are becoming ubiquitous; (3) software defined radio technologies have developed into an active area of research with commercial uses [24]; and (4) the Federal Communications Commission (FCC) has opened up "white spaces" spectrum to unlicensed devices.

Another wireless networking technology that is gaining popularity is community mesh networking [2] – a cost-effective mechanism for providing high speed wireless Internet connectivity to rural and urban communities where broadband wireless connectivity is unavailable or too expensive. Instead of dealing with mobility or minimizing power usage, the focus here is to increase the network capacity by reducing the interference [11]. Multi-radio multi-channel solutions have the potential to facilitate high throughput scalability in dense mesh network deployment scenarios to meet user needs.

In light of the above technological trends, several architectures and designs for dynamic spectrum access/sharing in cognitive radio networks [20, 24], and channel selection and routing in wireless mesh networks [10, 23, 6, 11] have been proposed. These proposals aim to mitigate the impact of harmful interference and thus improve overall network performance. For reasonable operation of large wireless mesh networks with nodes strewn over a wide area with heterogeneous policy constraints and traffic characteristics, we believe that a *one-size-fits-all* channel selection and routing protocol may be difficult, if not impossible, to find.

To address the above needs, this paper presents Policy-based Unified Multi-radio Architecture (PUMA) for agile mesh networking, a platform that aims to develop intelligent network protocols that simultaneously control parameters for dynamic (or agile) spectrum sensing and access, dynamic channel selection and medium access, and data routing with a goal of optimizing overall network performance.

PUMA aims to serve two important communities. For researchers, PUMA provides a common framework for rapid describing, evaluating, and comparing new channel selection and routing policies. For network operators, PUMA eases the process of implementing, configuring, and deploying mesh networks. Towards this goal, PUMA makes the following contributions:

Declarative channel selection and routing: In PUMA, channel selection policies are formulated as *constraint optimization problems* (COP) that are specified using the *PawLog* declarative language. The customizability of *PawLog* allows the providers a great degree of flexibility in the specification and enforcement of various local and global channel selection policies. These policy specifications are then compiled into efficient constraint solver [1] code for execution. Compared to traditional imperative alternatives, *PawLog* results in approximately $100X$ reduction in code size, and is easier to understand, debug and extend.

In addition to supporting policy specifications, PUMA integrates the constraint solver with a declarative networking engine [18]. This enables one to use *PawLog* to specify distributed COP programs that implement distributed channel selection protocols. The declarative networking engine can also be used for implementing multi-hop routing protocols [15, 18].

Distributed cross-layer protocol: By combining channel selection and routing within a common declarative framework, PUMA further enables new and interesting capabilities that are significantly easier to capture compared to tra-

ditional imperative approaches. As an example of such capabilities, we have developed a novel distributed cross-layer protocol that integrates and optimizes across channel selection and routing policy decisions. This protocol incorporates the traffic rate into consideration for channel selection, and improves route computation to jointly optimize for traffic load and channel diversity. As evidence of the advantages of declarative programming, the cross-optimized protocol requires minor modifications to existing declarative specifications to encode dependencies across route and channel selection policies.

Implementation and evaluation: We have developed a PUMA prototype using the Gecode [1] high-performance constraint solver and the RapidNet [5] declarative networking engine as building blocks. We have conducted extensive 802.11 wireless simulations and actual experimentation on the ORBIT [4] testbed. Our evaluation demonstrates that PUMA can implement a wide range of channel selection protocols that converge quickly and significantly outperform single-channel and identical channel assignment in terms of network throughput. Moreover, the cross-layer protocol significantly outperforms all other protocols, particularly when one incorporates traffic-aware policies into channel selection and routing.

2. OVERVIEW

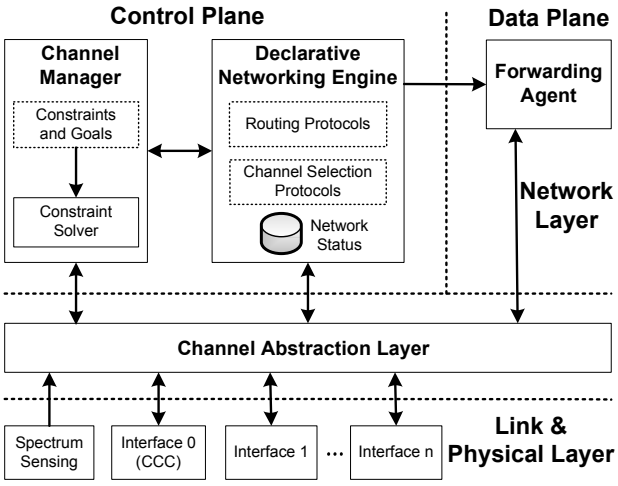

Figure 1: Components of a PUMA node. The components in dotted lines indicate *PawLog* inputs.

Figure 1 shows an overview of PUMA from the perspective of a single PUMA node.

Channel manager. The role of the channel manager is to assign available channels to wireless links to optimize a performance *goal* (e.g. minimize network interference, minimize the number of unique channels) while subjected to *constraints* (e.g. regional policies on spectrum usage [20], yield to primary users who own exclusive rights to certain spectrums in white space networks). In PUMA, we use the *PawLog* language for declaratively expressing goals and constraints as a *constraint optimization problem* (COP) [26]. These specifications are compiled into executables within Gecode constraint solver [1]. The channel manager takes as additional input *network status* information from the declarative networking engine, including network topology and the set of channels available to each node.

The channel manager can be deployed either in a centralized or distributed mode. In the centralized mode, all nodes send their local neighborhood and channel availability information to a centralized channel manager which performs channel assignment for the entire network. In the distributed mode, each node makes individual channel assignment decisions using its own solver, with only information gathered from neighbors within the vicinity.

Declarative networking engine. At the network layer, the RapidNet declarative networking engine [5] is deployed within the control plane to implement a variety of neighbor discovery and routing protocols also expressed in *PawLog*. These protocols can either be specified dynamically by the user, or pre-programmed as a library of declarative wireless routing protocols [15]. In addition, channel selection protocols enable nodes to exchange status information among themselves while performing channel assignment using the constraint solver. All network status computed by PUMA (e.g. neighbor discovery, routing, channel availability and assignments) are maintained and stored as RapidNet tables, and made available to other components via callbacks.

Note that routing protocols, channel selection goals and constraints, and channel selection protocols are all written in the same *PawLog* language. We will demonstrate later in this paper how one can leverage this unified declarative framework to encode policies that optimize across routing and channel selection.

Channel abstraction layer. Each PUMA node runs a number of multi-channel wireless radio devices (interfaces). Typically, the first interface operates on the *common control channel* (CCC), which is reserved for routing and channel selection protocol messages. A *spectrum sensing* [1] component is able to detect channels available for each interface by periodically scanning a wide range of spectrum. The set of available channel information is then made available to the channel manager through the channel abstraction layer [8], which interacts with multiple radios and presents upper layers with a multi-channel communication interface. In order for packets to be routed to neighbors using appropriate interface/channel, the output of the channel manager is then used to initialize the channel assignment table at the channel abstraction layer.

Forwarding agent. Finally, the output of declarative routing is a forwarding table (next-hop for each destination) used by the forwarding agent. Given a destination, the forwarding agent queries the channel abstraction layer to determine the corresponding interface/channel for the next-hop, and forwards the packet accordingly.

[1]Spectrum sensing is an orthogonal problem beyond the scope of this paper, where we focus on channel selection and routing.

3. DECLARATIVE CHANNEL SELECTION

In this section, we describe how to declaratively specify channel selection policies in the form of goals, constraints, and derivation rules. We first formulate channel selection as a constraint optimization problem (COP), followed by describing how these COP formulation can be expressed using *PawLog* and then compiled into efficient executions.

3.1 COP Formulation

A COP formulation takes as input a set of constraints, and attempts to find an *assignment* of *values* chosen from an input *domain* to a set of *variables* to satisfy the constraints under an optimization *goal*. The goal is typically expressed as a minimization over a cost function of the assignments. In the context of channel assignment, the variables are the channels to be assigned to each communication link, while the values are chosen from candidate channels available to each node. The goal in this case is to minimize the likelihood of interference among conflicting links, which maps into the well-known *graph-coloring* problem [13].

We consider the following example that avoids interference based on the *one-hop interference* model [28]. In this model, any two adjacent links are considered to interfere with each other if they both use channels whose frequency bands are closer than a certain threshold. The formulation is as follows:

Input domain and variables: Consider a network $G = (V, E)$, where there are nodes $V = \{1, 2, \ldots, N\}$ and edges $E \subseteq V \times V$. Each node x has an available set of candidate channels A_x to select from, and a set of channels P_x currently occupied by primary users within its vicinity. The number of interfaces of each node is i_x.

Optimization goal: For any two adjacent nodes $x, y \in V$, l_{xy} denotes the link between x and y. Channel assignment selects a channel c_{xy} for each link l_{xy} to meet the following optimization goal:

$$\min \sum_{l_{xy}, l_{xz} \in E, y \neq z} cost(c_{xy}, c_{xz}) \quad (1)$$

where $cost(c_{xy}, c_{xz})$ assigns a unit penalty if adjacent channel assignments c_{xy} and c_{xz} are separated by less than a specified frequency threshold $F_{mindiff}$:

$$cost(c_{xy}, c_{xz}) = \begin{cases} 1 & \text{if } |c_{xy} - c_{xz}| < F_{mindiff} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Constraints: The optimization goal has to be achieved under the following four constraints:

$$\forall l_{xy} \in E, c_{xy} \in A_x \quad (3)$$

$$\forall l_{xy} \in E, c_{xy} \notin P_x \quad (4)$$

$$\forall l_{xy} \in E, c_{xy} = c_{yx} \quad (5)$$

$$\forall x \in V, |\bigcup_{l_{xy} \in E} c_{xy}| \leq i_x \quad (6)$$

(3) ensures that each channel assignment c_{xy} is selected from the available channel domain A_x. (4) expresses the constraint that a node should not use channels currently occupied by primary users within its vicinity. (5) requires two adjacent nodes to communicate with each other using the same channel. (6) guarantees the number of assigned channels is no more than interfaces.

3.2 PawLog Specifications

Instead of hard-coding the COP formulation into a constraint solver, PUMA uses the *PawLog* language to concisely declare the formulation in the form of policy goals, rules and constraints. This results in orders of magnitude reduction in code size. The compact specifications further facilitate policy customization and enable us to rapidly explore and deploy a range of channel selection protocols.

As an example, the following *PawLog* program specifies the one-hop interference model COP formulation described in Section 3.1. This program requires only a handful of *PawLog* rules, and has a natural mapping to the mathematical formulation.

```
// goal declaration
goal minimize C in totalcost(C)

// variable declaration
var assignChannel(X,Y,C) forall link(X,Y)

// cost assignment rules
s1 cost(X,Y,Z,C) :- assignChannel(X,Y,C1),
   assignChannel(X,Z,C2), Y!=Z, C=1,
   |C1-C2|<F_mindiff.
s2 totalCost(COUNT<C>) :- cost(X,Y,Z,C).

// Input domain constraint for assignChannel
c1 assignChannel(X,Y,C) -> link(X,Y),
                           availChannel(X,C,F,St).

// primary user constraint
c2 assignChannel(X,Y,C) -> !primaryUser(X,C).

// channel symmetry constraint
c3 assignChannel(X,Y,C) -> assignChannel(Y,X,C).

// interface constraint
c4 uniqueChannel(X,Count) -> numInterface(X,K), Count<=K.
s3 uniqueChannel(X,UNIQUE<C>) :- assignChannel(X,Y,C).
```

In *PawLog*, two reserved keywords `goal` and `var` specify the goal and variables used by the constraint solver. *PawLog* constraints (`c1-c4`) are of the form F_1 `->` F_2, which denotes the logical meaning that whenever F_1 is true, then F_2 *must* also be true in order for the constraint not be violated. Derivation rules (`s1-s3`) of the form `p :- q1, q2, ..., qn.`, result in the derivation of `p`, whenever the rule body (`q1` and `q2` and `...` and `qn`) is true. Each term within a rule (e.g. `q1`, `q2`) is typically referred to as a *predicate*, and the corresponding data output obtained during rule execution are referred to as *tuples*. The derivation rules are based on *Network Datalog* [18], a recursive query language used in declarative networking for computing network graph properties. *PawLog* can hence be viewed as a superset of *Network Datalog*, with additional constructs specific to constraint solving (e.g. `goal`, `var`, and constraint rules). We next describe in detail the input/output to the program, goal and constraints:

Input tables: The above program takes as two input tables `link(X,Y)` and `availChannel(X,C,F,St)`. As de-

scribed in Section 2, the `link` table stores the gathered network topology information, and the `availChannel` table is supplied by the channel abstraction layer via known spectrum sensing mechanisms, where each entry denotes that node `X` has an available channel `C` with frequency `F` and signal strength `St`.

Output tables: The solver outputs `assignChannel(X,Y,C)` table, where each entry indicates channel `C` is used for communication between `X` and `Y`. The channel abstraction layer uses this information to select an unused interface to run on channel `C`, and then updates its internal state that stores the mapping from neighbors to interfaces. This information will be used by the forwarding agent to ensure that all messages forwarded to neighbors are directed to the selected interface.

Optimization goal: The goal in this case is to minimize the cost attribute `C` in `totalCost`, while assigning channel variables `assignChannel` for all communication links. Rule `s1` sets cost `C` to 1 for each `cost(X,Y,Z,C)` tuple if the chosen channels that `X` uses to communicate with adjacent nodes `Y` and `Z` are interfering. Rule `s2` counts the number of interfering channels among adjacent links in the entire network, and stores the result in `totalCost`.

Constraints: Policy *constraints* are used to remove illegal channel assignments. These constraints can be globally applied to all nodes, or customized at the node-level. The constraints `c1-c4` encode the four constraints introduced in COP formulation in Section 3.1. Constraint `c1` restricts the domain of `assignChannel(X,Y,C)` to only valid channel assignments for existing links `link(X,Y)` and ensures that only available channels are considered. Constraint `c2` applies to the input `availChannel` table, and states that a channel `C` at node `X` is only available, if there does not exist a primary user within the vicinity of `X`. Constraint `c3` enforces channel symmetry on the output `assignChannel` table. Constraint `c4` requires that nodes must use at most K unique channels, where K is the number of usable interfaces. The number of unique channels is derived in rule `s3` using aggregate keyword `UNIQUE`.

3.3 Policy Customizations

One of the advantages of declarative programming is the ease of customization, which can often be achieved with only minor modifications to existing policies. We illustrate some examples here.

In some wireless deployments, e.g. IEEE 802.11, the *two-hop interference model* [28] is often considered a more accurate measurement of interference. This model considers interference that results from any two links using similar channels within two hops of each other. The two-hop interference model requires minor modifications to rule `s1` as follows:

```
s1a cost(X,Y,Z,W,C) :- assignChannel(X,Y,C1), link(Y,Z),
    assignChannel(Z,W,C2), X!=Z, Y!=Z, X!=W, C=1,
    |C1-C2|<F_mindiff.
```

The above rule considers four adjacent nodes `X`, `Y`, `Z`, and `W`, and assigns a cost of 1 to node `X`'s channel assignment with `Y` (`assignChannel(X,Y,C1)`), if there exists a neighbor `Z` of `Y` that is currently using channel `C2` that interferes with `C1` to communicate with a node other than `Y`. The above policy requires only adding one additional `link(X,Y)` predicate, demonstrating the customizability of *PawLog*. Together with the original rule `s1`, one can assign costs to both one-hop and two-hop interference models. Furthermore, one can easily generalize to K-hop interference model using a recursively defined rule. Interference models based on other indicators, e.g. Received Signal Strength Indication (RSSI), are as well succinctly expressible in *PawLog*.

In addition, PUMA can flexibly declare more constraints, e.g., impose regional policies on spectrum usage; avoid channels that have low SNR (a straightforward filter condition on `availChannel` table); ensure channel diversity along each path (by having the cost assignment take into account of interference along each best path); minimize the number of unique channels in a network while ensuring no link conflicts [14] (by making the cost function a "hard constraint" which incurs infinite cost if violated).

3.4 PawLog Compilation

PawLog programs are compiled into executions within the Gecode [1] solver and the RapidNet declarative networking engine [5]. Gecode is used for high-performance constraint solving, while RapidNet is used for table materialization, rule execution, and distributed implementation of channel selection protocols (Section 4). In a typical deployment, RapidNet runs the channel selection and routing protocols at each node, and invokes Gecode's COP modules when a channel assignment is required.

Our compilation process maps *PawLog*'s `goal`, `var`, and constraints into equivalent COP primitives in Gecode. Prior to running the COP in Gecode, the generated code loads in the appropriate input data from RapidNet, and then stores the output results (channel assignments) in RapidNet after COP execution. We note that this compilation process is generic and can be applied to other solvers as well.

The more interesting aspect of our compilation process is the interplay between Gecode solver and RapidNet declarative networking engine. The derivation rules of *PawLog* programs are executed using database operators, such as joins (variable matching in rule body), aggregation (e.g. COUNT, UNIQUE), selection filters, rule head renaming, etc. For efficiency and code reuse purpose, these rule executions are offloaded from the solver to RapidNet's query engine. The solver adopts the standard branch-and-bound searching approach to solve the optimization while exploring the space of variables under constraints. In cases where the rule body contains solver variables (e.g. rule `s1`), instead of running these rules within RapidNet, we perform a rule rewrite process that transforms derivation rules into solver constraints to prune the search space.

Finally, the declarative networking engine is also used for executing policy rules whose body predicates span across multiple nodes (Section 4.2). All derivation rules executed in RapidNet are done in a continuous, long-running fashion, where rule head tuples are continuously updated (inserted or

deleted) in an incremental fashion [17] as the body predicates are updated. As we show in subsequent sections, this allows us to incrementally re-optimize channel selection as the underlying network topology changes.

4. CHANNEL SELECTION PROTOCOLS

Given the declarative channel selection policies introduced in the previous section, we next describe how these policies can be realized in an actual deployment by adding additional *PawLog* rules. Specifically, we present a centralized and a distributed channel selection protocol implemented using PUMA. In both protocols, we consider channel selection to be carried out separately from routing. In Section 5, we relax this requirement, and take PUMA's approach one step forward by presenting a novel distributed traffic-aware protocol that optimizes across route and channel selection policies.

4.1 Centralized Channel Selection

In centralized channel selection [23, 7], the channel manager is deployed on a single node in the network. Typically, this node is a designated server node, or is chosen among peers via a separate leader election protocol.

Due to space constraints, rather than present the entire *PawLog* rules, we provide a high-level intuition on how the *PawLog* program is formulated. The centralized manager collects the network status information from each node in the network – this includes their neighborhood information, available channels, and any additional local policies. The network status information can be collected using link-state dissemination and its variants expressible also as declarative rules [15]. Alternatively, if a route to the centralized solver has already been computed, each node can forward the status information via the CCC directly to the centralized solver.

After gathering network status information, the centralized channel manager has access to the entire network topology (`link` table) and available channels (`availChannel` table). It then uses the solver to execute the policy rules described in Section 3 to generate channel assignments `assignChannel(@X,Y,C)` for each node `X`. This information is then propagated to each node `X` to set its local channel to neighbor `Y` accordingly. Here, the *location specifier* `@` is a common symbol used in declarative networking [18], denoting the source location of each corresponding tuple. It is essential for ensuring that each derived `assignChannel(@X,Y,C)` tuple is sent to the appropriate node `X`.

Given that graph coloring is an NP-hard problem, to find the solution in reasonable time one approximation method we have explored is a *divide-and-conquer* strategy. The basic idea is to divide the whole network into roughly equal-sized subnetworks (we use a heuristic breadth-first search to partition the network), and have the solver perform channel selection over each smaller subnetwork. Interestingly, this division process requires minimal changes to the *PawLog* policy rules, simply by partitioning the input tables into smaller ones. Once the channels are assigned to individual subnetworks, the remaining links (or bridges) connecting the subnetworks together are assigned channels that minimize the overall interference cost. If no new channels are available (i.e. all interfaces have already been assigned channels during the earlier subnetwork optimization phase), the CCC is used for communication as a fallback.

4.2 Distributed Channel Selection

We next demonstrate PUMA's ability to implement distributed channel selection. Distributed channel selection provides approximations to the optimal centralized solution, and hence scales better for large networks. Moreover, it has the added advantages of not introducing single points of failure and is amenable to incremental computations as the network topology changes. Our example here is based on a variant of distributed greedy protocol proposed in [25]. This example highlights PUMA's ability to support *distributed COP* computations, where nodes compute channel assignments (a COP computation) based on local neighborhood information, and then exchange channel assignments with neighbors to perform further COP computations. This distributed approach is achieved by PUMA's use of a declarative networking engine in conjunction with a constraint solver.

The protocol works as follows. Periodically, each node randomly selects one of its links (*link selection*) to start a *channel negotiation* process with its neighbor. To avoid conflicting channel assignments, for any given `link(i,j)`, the link selection protocol selects the node with the larger identifier (or address) to carry out the subsequent channel negotiation process. Once a link is selected for channel assignment, the negotiation process solves a *local* COP and assigns a channel such that interference is minimized. In case there are several solutions with minimum interference cost, the solve randomly picks one. The new channel-to-link assignment is then propagated to immediate neighbors.

The following *PawLog* program implements the local COP operation at every node `X` for performing channel assignment. The output of the program sets the channel `assignChannel(@X,Y,C)` for one of its links `link(@X,Y)` (chosen for the current channel negotiation process) based on the two-hop interference model:

```
goal minimize C in totalcost(C)
var assignChannel(@X,Y) forall eSetLinkChannel(@X,Y)

// trigger the start of the solver
d1 eStartSolver(@X) :- eSetLinkChannel(@X,Y).

// two-hop assignments
d2 twoHopChannels(@X,Y,Z,C) :- link(@X,Y),
   assignChannel(@Y,Z,C).

// propagate channels to ensure symmetry
d3 assignChannel(@Y,X,C) :- assignChannel(@X,Y,C).

// cost assignment for two-hop interference model
ds1 cost(@X,Y,Z,W,C) :-  twoHopChannels(@X,Z,W,C1),
    assignChannel(@X,Y,C2), W!=X, Z!=Y, W!=Y, C=1,
    |C1-C2|<F_mindiff.

// aggregate the cost
ds2 totalCost(@X,COUNT<C>) :- cost(@X,Y,Z,W,C).

// Input domain constraint for assignChannel
dc1 assignChannel(@X,Y,C) -> link(@X,Y),
    availChannel(@X,C,F,St1), availChannel(@Y,C,F,St2).

// primary user constraint
```

```
dc2 availChannel(@X,C,F,St) -> !primaryUser(@X,C).
```

The event that triggers the solver execution of the above program is denoted by `eSetLinkChannel(@X,Y)`. This event is periodically generated as part of the link negotiation process, and `Y` denotes the neighbor chosen for the current negotiation process. The distributed program is similar to the centralized equivalent presented in Section 4.1, with the following differences:

Localized COP: While the centralized channel selection searches for all combinations of channel assignments for all nodes, the distributed equivalent restricts channel selection to a single link one at a time, where the selected link is represented by `eSetLinkChannel(@X,Y)` based on the negotiation process. For this particular link, the COP execution takes as input its local neighbor set (`link`), the available channels (`availChannel`), and all currently assigned channels (`assignChannel`) for itself and nodes in the local neighborhood. This means that the COP execution is an approximation based on local information gathered from a node's neighborhood.

Distributed execution: The use of location specifier `@` enables one to naturally capture constraints and dependencies involving nearby neighbors. Rules `d2` enables a node `X` to collect the current set of channel assignments for itself and its immediate neighbors `Y`. In executing the channel selection for the current link, constraint `dc1` limits the channel assignment for `link(@X,Y)` to only channels common to both `X` and `Y`. Once a channel is set at node `X`, the channel assignment is immediately propagated to neighbor `Y`, hence resulting in symmetric channel assignments (rule `d3`).

In essence, one can view the distributed protocol as a series of per-node COP carried out using each node's constraint solver. The channel negotiation process is repeated at each node periodically until all links have been assigned a channel. Each channel negotiation process will use the link channel assignments computed in previous rounds in order to determine the channel assignment for the next link.

In situations when the constraint solver returns no solution, the link is assigned to use the CCC as a fallback. The complexity of this protocol depends upon the maximum node degree, since each node at most needs to perform m rounds of channel negotiation, where m is the node degree.

Incremental updates: Link and node dynamics are easily captured in *PawLog*, via a technique known as *incremental view maintenance* [17], that essentially stores the rule results and incrementally updates the results as the rule body predicates are updated. This avoids having to recompute a rule from scratch whenever the inputs to the rule change. For example, in rule `d2`, `twoHopChannels` tuples are updated whenever `link` and `assignChannel` are updated. This results in changes to the `cost` and `totalCost` values, which will further result in new channel assignment values when the solver is next executed for the new link.

5. CROSS-LAYER OPTIMIZATIONS

Using the distributed channel selection protocol as a basic building block, we present a distributed cross-layer protocol that optimizes across channel selection and routing. While similar cross-layer optimizations have received attention in a centralized context [23], our proposed protocol (to our best knowledge) is the first to be implemented in a fully distributed fashion. PUMA's use of declarative networking enables us to compactly and naturally realize this distributed protocol, requiring minimal modifications to the *PawLog* rules we presented in Section 4.2. We further discuss enhancements to the route selection metric to more effectively take into account channel diversity and traffic load.

Figure 2 outlines the steps taken by the distributed cross-layer protocol. Each box indicates a step (*component*) which encapsulates a set of *PawLog* rules. The output of each component can be directly used as input to the next component, simply by having rules in the next component be defined in terms of the output from the previous component.

We provide a brief description of each component:

Distributed channel selection: The first component is *Distributed Channel Selection*, which reuses the set of rules presented in Section 4.2. This process is usually started in the initial phase of bootstrapping channel assignments.

Link-state update propagation: Following channel selection, a set of *PawLog* rules are used for implementing a flood-based propagation of link-state updates (LSUs), using either traditional link-state dissemination or more scalable variants (e.g. OLSR [9]). The LSUs in this case include the neighborhood set of each node, and also the assigned channels for each link as computed in the previous step.

Route computation: Based on the LSUs, a *Route Computation* component executes the next step to compute best paths based on the WCETT [11] metric. WCETT is a path-based metric, resulting in the selection of a route with maximum path channel diversity (Appendix A gives a brief description of WCETT).

Link traffic estimation: Based on the computed routes, another set of *PawLog* rules implement the *Link Traffic Estimation* component, which estimates the expected traffic load for each link. Consider two nodes i and j. The traffic on `link(i,j)` is estimated as $S_i \times P_{ij}$ where S_i is the data sending rate of node i, and P_{ij} is the probability that the link appears along selected best paths of node i. For the initial bootstrapping stage, we assume P is same for each link. The aggregate traffic between i and j is then calculated by summing the traffic from i to j and from j to i. Link traffic estimation can be incrementally updated as routes are updated, or as the sending rate of nodes changes at runtime. The link traffic estimation is subsequently used as an *interference cost* input to another round of *Distributed Channel Selection*, making the channel selection process traffic-aware.

The above four components consist of 25, 11, 4 and 7 *PawLog* rules, respectively. *PawLog* specifications of the cross-layer protocol result in significantly fewer code (and less code complexity) compared to alternative imperative im-

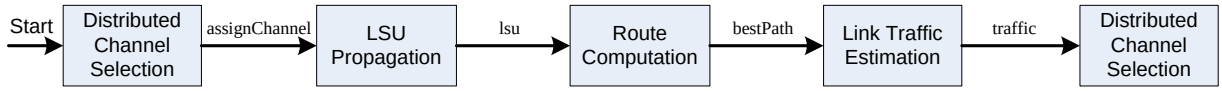

Figure 2: Components in distributed traffic-aware cross-layer protocol

plementations (See Section 7 for code size comparison).

5.1 Example *PawLog* Program

To highlight the interactions across these components, we revisit the distributed channel selection protocol in Section 4.2, and show a fragment of the modified *PawLog* program:

```
// LSU includes channel information
ls1a lsu(@X,X,Y,C,X,Ch) :- link(@X,Y,C),
     assignChannel(@X,Y,Ch).
ls2a lsu(@M,X,Y,C,Z,Ch) :- link(@Z,M,C1),
     lsu(@Z,X,Y,C,W,Ch), M!=W.

// two-hop traffic
d2a twoHopTraffic(@X,Y,Z,T) :- link(@X,Y),
    traffic(@Y,Z,T).

// two hop interference model
ds1a cost(@X,Y,Z,W,C) :- twoHopChannels(@X,Z,W,C1),
     assignChannel(@X,Y,C2), W!=X, Z!=Y, W!=Y,
     twoHopTraffic(@X,Z,W,T1), twoHopTraffic(@X,W,Z,T2),
     C=T1+T2, |C1-C2|<F_mindiff.
```

Rules `ls1a-ls2a` customize the link-state propagation rules [15], by additionally propagating the assigned channel `Ch` attribute for each `link(@X,Y,C)`. Rule `ds1a` replaces the earlier rule `ds1` in Section 4.2, where the cost of channel assignment is now the aggregate traffic load, which sums up the bi-directional traffic on all adjacent interfering links, given the two-hop interference model.

The rules above together with the ones for *Route Computation* construct a *mutual dependency* naturally captured via *PawLog*: in rule `ds1a`, the constraint solver will use the cost of aggregate traffic load derived from route selection to determine channel assignments; these assignments are subsequently propagated via rules `ls1a-ls2a` for channel diversity based routing using WCETT metric.

Based on Figure 2, the cross-layer optimization process stops when the second round of channel selection completes (with some channel assignments possibly refined). An alternative (fancier) approach is to consider an indefinite feedback loop in which the process of channel and route selection is repeatedly co-optimized until some theoretically sound stopping criteria is reached, e.g. when the network interference is below certain level. While these extensions are feasible to be explored and implemented in PUMA, repeated loops result in longer convergence time and yield diminishing returns. A detailed theoretical analysis of the stopping criteria is an interesting avenue for future work.

The above program works under the assumption that network traffic is relatively stable over the period of channel selection and routing. Given that wireless mesh networks usually have relatively fixed traffic patterns for hours [23], this assumption normally holds. In the event that traffic patterns keep fluctuating, we can easily modify the *PawLog* rules to fallback to a traffic agnostic policy (like the distributed protocol in Section 4.2).

5.2 Traffic-aware WCETT Enhancement

The WCETT [11] metric is based on *Expected Transmission Time* (ETT), which depends on link capacity and packet loss rate due to signal strength degradation. In practice, ETT is also affected by link traffic. The higher the traffic load, the more likely ETT will increase due to collisions and congestion related delays. To avoid routing along links with high traffic, we propose the *Traffic-aware WCETT*, which multiplies the original ETT value for each link by estimated traffic that traverses the link. Achieving this requires minimal changes to our *PawLog* rules, since link traffic estimation is already stored in the `traffic` table. Our evaluation results in Section 6 demonstrate that using the traffic-aware enhancement to WCETT results in further performance improvements over a regular distributed channel selection protocol, in addition to the improvements that are attributed to the cross-layer optimizations presented earlier in this section.

6. EVALUATION

We have developed a prototype of PUMA using the RapidNet declarative networking engine [5] and the Gecode [1] constraint solver. PUMA takes as input channel selecting and routing policies written in *PawLog*, and then generates RapidNet and Gecode C++ code using the compilation process described in Section 3.4.

Our PUMA platform provides two execution modes: (1) The *simulation mode* uses ns-3 [3] as its simulated network layer. ns-3 is an emerging discrete event-driven simulator intended as an eventual replacement for the popular ns-2 simulator. ns-3 emulates all layers of the network stack, supporting configurable loss, packet queuing, network topology models, and the IEEE 802.11b PHY+MAC model; (2) The *implementation mode* runs the same PUMA instances as in simulation, but instead of having the instances communicate via a simulated network, uses actual sockets to allow PUMA instances deployed on different physical nodes to communicate with each other. This allows us to perform realistic performance evaluation on the ORBIT [4] testbed.

In both modes, PUMA supports multi-radio multi-channel capabilities via our implementation of the *channel abstraction layer* [8] described in Section 2. Simulations enable us to evaluate the performance of various protocols in a controlled environment where we can vary network topology, traffic patterns, number of interfaces and available channels. This complements our ORBIT testbed evaluation.

In the absence of a publicly available imperative platform with capabilities (such as unified centralized/distributed channel selection and routing) comparable to PUMA, our evaluation focuses on validating PUMA's flexibility and efficiency. Specifically, we evaluate the PUMA platform along the following dimensions.

***PawLog* execution:** Our first evaluation goal is to quantify

the overhead of resources required for channel selection and routing. For each *PawLog* program executed using PUMA, we measure the solver execution time for centralized channel selection, as well as per-node communication overhead and convergence time for distributed protocols. The objective here is to validate that *PawLog* can be deployed at reasonably low overhead.

Channel selection and routing policies. Our second evaluation goal is to evaluate the policies and protocols used for channel and route selection. We injected packets into PUMA nodes with increasing sending rate, and then measure the *aggregate network throughput* defined in terms of network-wide aggregate data packet transmissions that are successfully received by destination nodes. In our experiments, we either fix the channel selection policy and vary the protocols (e.g. centralized vs distributed), or fix the channel selection protocol and vary the policies (e.g. one-hop vs two-hop interference).

***PawLog* language.** In our third evaluation goal, we evaluate the flexibility of *PawLog* in supporting a wide range of channel selection and routing policies. We further provide evidence on the compactness of *PawLog*, by comparing the lines of code in *PawLog* and the generated C++ code.

6.1 Experimental Setup

We evaluate the following channel selection protocols: *Centralized* (Section 4.1), *Distributed* (Section 4.2), *Cross-layer* (Section 5), and *Cross-layer (E)* (Section 5.2). Recall that the enhanced cross-layer protocol refines WCETT by taking link traffic load into route computation. For each protocol, channel selection combines the use of one-hop and two-hop interference models, by attempting to solve a COP that minimizes the number of conflicting links under either models.

As a basis of comparison, we consider two baselines *1-Interface* and *Identical-Ch*. In *1-Interface* all nodes communicate with each other using one interface and hence a common channel. In *Identical-Ch* [11], the same set of channels are assigned to the interfaces of every node (e.g. channel 1 to the first interface, channel 2 to the second), and a centralized constraint solver then assigns each link to use one of these interfaces. All of these protocols use the WCETT metric for routing, and runs a declarative link-state protocol [15].

In all our setups, nodes utilize multiple interfaces consisting of homogeneous multi-channel radios. We limit the set of usable channels to "orthogonal channels", i.e. channels with sufficient frequency gap between them to incur minimal or no interference when active in each other's vicinity. This limits interference to situations where nearby links use the same channel.

6.2 Small Network Simulations

Our first experiment consists of a small network of 12 nodes randomly located in a $450m \times 450m$ arena. Although channel assignment is NP-hard in complexity, the small network size enables the centralized solver to achieve an optimal solution in reasonable time. This allows us to compare the optimal solution with other protocols.

In this setup, the transmission range of each node is approximately $100m$, and all nodes communicate using ns-3's 802.11b PHY+MAC model. Each node has an average degree of 4, and is equipped with one interface reserved for CCC, and two additional data interfaces with 4 orthogonal channels each. All interfaces have capacity of $11.0Mbps$. By default our simulations do not use RTS/CTS among nodes, but permit up to 3 retries at the MAC layer to transmit each packet.

Convergence time. *Centralized* requires less than 10 seconds to perform channel selection on a Intel Quad core 2.33GHz PC with 4GB RAM running Ubuntu. The execution time is dominated by the computation overhead of the Gecode solver. The distributed protocols converge quickly as well – at 40 seconds and 80 seconds respectively for *Distributed* and *Cross-layer*. These convergence times represent the cost of performing channel selection *from scratch* on the entire network until all links have been assigned channels.

Convergence in the distributed case is dominated by the network degree and the periodic timers between each individual link channel negotiation. Since the solver computation only requires input channel information within a node's neighborhood, each per-link COP computation during negotiation is highly efficient and completes within 200ms. In the steady state after the initial channel assignments, as topology changes, the distributed protocols require only incremental recomputation by performing a channel negotiation for each modified link. On the other hand, the centralized approach requires a complete solver recomputation (for the entire network) even when only one link has changed.

Aggregate network throughput. Figure 3 shows the aggregate network throughput for five protocols as the traffic load increases. The traffic load consists of UDP data packets sent from random sources to random destinations. We observe that the throughput for all protocols first increases linearly as expected, then becomes sub-linear, and finally flattens when the network is saturated.

The baseline protocol *1-Interface* is the first to saturate (at an aggregate traffic load of $3.0Mbps$), followed by *Identical-Ch* (at $6.5Mbps$). *Centralized* (the optimal channel selection solution) consistently outperforms all other protocols at traffic loads below $8.0Mbps$. At higher traffic loads, *Cross-layer* outperforms even the *Centralized* solution. This is because *Cross-layer* is able to select channels and routes to intelligently bypass congested hot spots, an optimization that cannot be easily achieved when routing and channel selection decisions are carried out separately. In Figure 3, we omit *Cross-layer (E)*, since its performance is similar to *Cross-layer* for small networks.

Bandwidth utilization. *Distributed* and *Cross-layer* are both bandwidth efficient, requiring only per-node average bandwidth utilization of $7.98Kbps$ and $8.55Kbps$ respectively for computing channel selection from scratch.

6.3 Large Network Simulations

We repeat the same experimental setup as Section 6.2, but

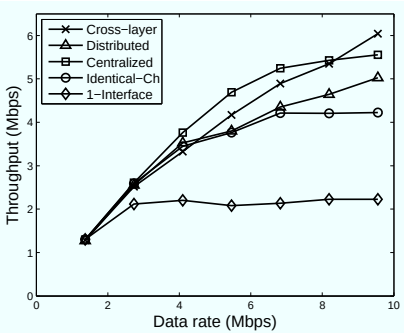

Figure 3: Aggregate network throughput (12 nodes).

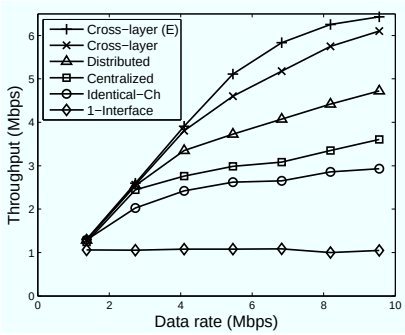

Figure 4: Aggregate network throughput (30 nodes).

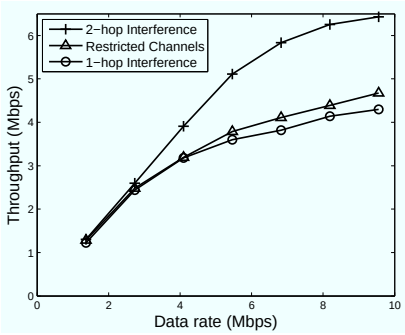

Figure 5: Aggregate network throughput (varying policies).

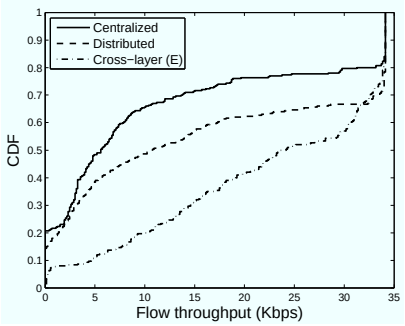

Figure 6: CDF of flow throughput (individual flow model).

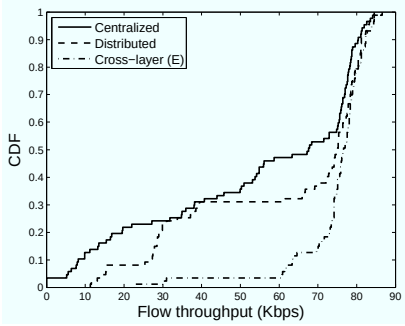

Figure 7: CDF of flow throughput (gateway traffic model).

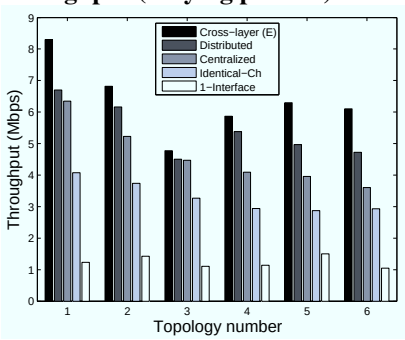

Figure 8: Aggregate network throughput (varying topologies).

utilize a larger mesh network consisting of 30 nodes randomly located within a $600m \times 600m$ arena with an average node degree of 5. We configure each node to use three data interfaces, where each interface has an available set of 8 orthogonal channels.

Convergence time. Given the complexity of channel assignment and the size of the network, the *Centralized* solution that searches for the optimal solution (not surprisingly) require days to complete. As a result, we utilize the *divide-and-conquer* approximation introduced in Section 4.1, which divides the network into roughly equal-sized subnetworks. The smaller the size of each subnetwork, the faster the solver terminates, but the less optimal the solution is. For a subnetwork of size 7, PUMA's solver is able to generate channel assignments within 30 seconds. For all distributed protocols, the convergence time is similar to that of the earlier 12-node experiment. This is because the convergence time for channel selection for these protocols is determined by node degree and link negotiation interval, not the network size.

Aggregate network throughput. Figure 4 shows the aggregate network throughput for all protocols. We observe that the scalability trend is similar to the 12 nodes setup (Figure 3), but with some key differences: (1) *Distributed* consistently has higher throughput than *Centralized*, suggesting that while both are approximations to the optimal solution, the *divide-and-conquer* strategy is not as effective as a purely greedy approach that assigns link channels one at a time; (2) *Cross-layer*'s relative improvements over the other protocols are even more apparent in a large network, achieving an average 24.7% higher throughput compared to *Distributed*, and a $1.9X$ and $4.8X$ improvement over *Identical-Ch* and *1-Interface* respectively; (3) the best performing protocol is *Cross-layer (E)*, which has an improved throughput of 9.5% over the basic *Cross-layer*.

Bandwidth utilization. *Distributed*, *Cross-layer*, and *Cross-layer (E)* incur low per-node average bandwidth utilization of $12.57Kbps$, $12.64Kbps$ and $12.80Kbps$, respectively. The performance numbers for network throughput and bandwidth utilization suggest that per-node overhead is low. Moreover, as network size increases, distributed protocols are a more attractive option compared to a centralized strategy in generating good channel assignments within reasonable communication overhead and convergence time.

6.4 Policy Customization

In all our previous experiments, we have fixed the channel selection policy but vary the mechanisms (e.g. centralized vs distributed). Given the same 30-node setup, Figure 5 highlights the capabilities of PUMA to handle policy variations with minor changes to the input *PawLog* policy rules.

Specifically, we fix the protocol to be *Cross-layer (E)*, and then vary the policies in two ways. First, *Restricted Channels* reduces the number of available channels for each node by an average of 20%. This emulates the situation where some channels are no longer available due to external factors, e.g. decreased signal strength, the presence of primary users, or geographical spectrum usage limits. Second, *1-hop Interference* uses a different cost assignment function to consider only one-hop interference. As a basis of comparison, *2-hop Interference* shows our original channel selection policy used in prior experiments.

We observe that for *Restricted Channels*, the throughput decreases by 35.9%. With the additional use of one-hop interference model, the throughput further reduces by an average of 6.9%, indicating that the two-hop interference model does a better job in ensuring channel diversity.

6.5 Varying Traffic and Topology

Using the 30 node setup, we examine the sensitivity of our results by varying traffic patterns and network topologies.

Individual flow model. Here, instead of using a random traffic model, we selected 300 random source/destination pairs, and then generate a steady stream of packets at a bidirectional sending rate of $17.00Kbps$ (in each direction) between each pair routed along the computed best path. Figure 6 shows the CDF of network throughput breakdown by individual network flows. We observe that under flow traffic, *Cross-layer (E)* achieves the best performance, followed by *Distributed*, and then *Centralized*. We omit presenting *1-Interface* and *Identical-Ch* in the CDF, since their throughput is far below *Centralized*.

Gateway traffic model. Figure 7 shows a similar CDF under a different *gateway* traffic model. Instead of selecting random sources/destinations to construct the flows, the gateway model limits the flows between three designated gateway nodes and randomly selected other nodes. This creates a well-studied traffic scenario [10, 25, 22] where gateway nodes are receiving/disseminating data from/to other nodes in the network. We observe that the relative performance differences (in terms of flow throughput) among protocols are consistent with our earlier observations.

Different topologies. To analyze the sensitivity of different topologies, Figure 8 repeats the same experiment as Figure 4 with an aggregate data rate of $9.6Mbps$ between random sources and destinations. Across six different topologies experimented, we measure the aggregate network throughput, and again observe similar trends as before: *Cross-layer (E)* achieves the highest throughput, followed by *Distributed*, *Centralized*, *Identical-Ch*, and finally *1-Interface*.

Heterogeneous traffic. Finally, we conduct experiments where the sending rate of nodes vary based on the normal distribution. Figure 9 shows the performance for all protocols for different normal distributions, from the least heterogeneous (on the left as pattern 1) to the most heterogeneous (on the right as pattern 3). The aggregate network-wide sending rate is fixed at $6.8Mbps$. For all three traffic patterns, both *Cross-layer* and *Cross-layer (E)* consistently outperform other protocols. Moreover, *Cross-layer (E)* yields an 11.1% throughput improvement (on average) over *Cross-layer*. Interestingly, since the other four protocols are already performing poorly, they are not affected much by the degree of traffic heterogeneity.

6.6 ORBIT Testbed Results

Our final set of experiments are carried out on the ORBIT [4] testbed, a wireless testbed that consist of machines arranged in a grid communicate with each other using 802.11. Each ORBIT node is equipped with 1 GhZ VIA Nehemiah processors, 64KB cache and 512MB RAM. We selected 30 ORBIT nodes in a $8m \times 5m$ grid to execute one PUMA instance each. Each of these 30 nodes utilizes two Atheros AR5212-based 802.11 a/b/g cards as their data interfaces. By default, RTS/CTS is not used, and nodes are configured without retries.

ORBIT is one of the publicly available wireless testbeds for carrying out large-scale wireless experiments. This testbed allows us to validate results obtained in simulations. One current limitation of ORBIT is that given that the maximum distance between any two nodes in our experiment is about 9.4 meters, all nodes can hear the transmission signals from all other nodes.

To mitigate this issue, we make the following changes: (1) reduce the transmission power of all nodes to $1dBm$; (2) utilize `iptables` to filter packets at the MAC layer to emulate a grid topology where each node only receives messages from its designated neighbors within 1-2 meters range; (3) nodes communicate using 802.11a/g with $54Mbps$ capacity, which we have found to result in higher saturation bandwidth on ORBIT compared to 802.11b. In total, we have 10 orthogonal channels (7 from 802.11a and 3 from 802.11g) to assign to the two data interfaces. Even with these mitigation techniques, we note that the resulting network is physically a fully-connected mesh since any two nodes can still receive signals from each other. This limitation of ORBIT impacts the absolute throughput across all protocols experimented.

The first experiment evaluates our protocols using the random traffic model similar to previous simulations. After channel assignments and routes are established, packets are routed from randomly selected sources to destinations along the best paths. Due to ORBIT's configuration as a fully-connected mesh, the utility of traffic-aware routing is limited. We hence focus on comparing the regular *Cross-layer* with other schemes. Figure 10 shows the aggregate network throughput as the offered load increases until saturation is reached for most protocols.

We make the following observations. First, the comparative differences across protocols are consistent with our earlier simulation results. The best performing protocol is *Cross-layer* which has the highest aggregate throughput, followed by *Distributed* and *Centralized* (with divide-and-conquer approximation).

Second, *1-Interface* and *Identical-Ch* have reductions in throughput at high data rates greater than $8.0Mbps$. We attribute this to congestion-related high packet losses due to increased interferences at high data rates. These losses are more apparent for protocols with limited channel diversity. *Cross-layer* on the other hand avoids throughput degradation through a better choice in route and channel selection.

Third, we observe that PUMA is able to handle high rates of traffic. For instance, when data is injected into the network at a rate of $7.5Mbps$, the *Cross-layer* protocol is able to forward packets efficiently with only a loss rate of 14.7%. Given that each packet traverses 3-4 hops from source to

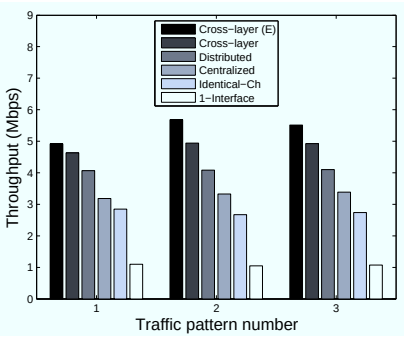

Figure 9: Aggregate network throughput (varying traffic).

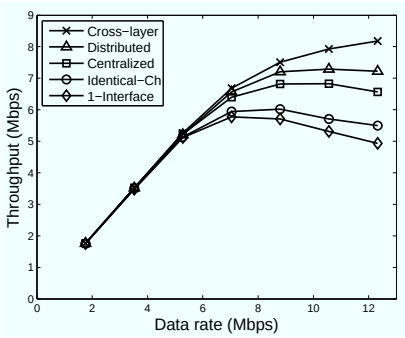

Figure 10: Aggregate ORBIT throughput (30 nodes).

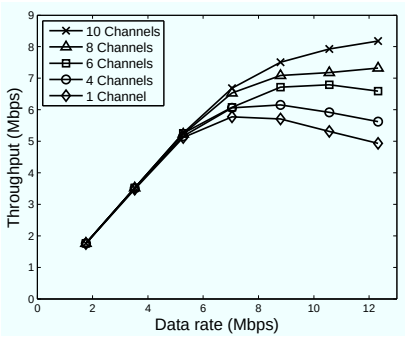

Figure 11: Aggregate ORBIT throughput (varying #channels).

destination, this translates to about $1.0Mbps$ per-node bandwidth. Given ORBIT's fully-connected mesh, these performance results should be viewed as a lower bound.

In our second experiment, we execute *Cross-layer* using the same setup as above, but limits the number of available channels to 10, 8, 6, 4, and 1 (fall back to *1-Interface*). This setup emulates the situation where some channels are unavailable due to policy constraints same as simulations in Section 6.4. Figure 11 shows whenever more channels are available, PUMA is able to leverage the increased channel availability to reduce interference and hence results in increased throughput.

7. DISCUSSION

We discuss our experimental results by revisiting our evaluation criteria used in Section 6:

***PawLog* execution:** PUMA is able to perform channel selection and routing in a bandwidth-efficient manner with fast convergence time, and handle high traffic rates before reaching interference-induced saturation.

Channel selection and routing policies. This paper is one of the first comprehensive attempts at experimentally evaluating a wide range of channel selection protocols in both simulations and actual testbed. Centralized and distributed channel selection and routing protocols implemented in PUMA significantly outperform single-channel and identical channel assignment solutions. The relative differences and scalability trends of these protocols are consistent with what one would expect in imperative implementations.

Our proposed cross-layer protocol (particularly when with the enhancement of traffic-aware routing) exhibits the best overall performance in terms of high throughput and low loss rate, and especially in large networks, significantly outperforms other protocols.

Flexibility of *PawLog*. Our evaluation validates *PawLog*'s flexibility in supporting various channel selection protocols (e.g. centralized, distributed, cross-layer) and policies (e.g. one-hop vs two-hop interference models, restricted channels). *PawLog*'s high level abstractions make it extremely easy to encode novel policies, such as the cross-layer strategy that optimizes across channel selection and routing. This is a clear evidence on the advantages of a declarative approach, which allows us to rapidly prototype and evaluate varying protocols and policies.

Protocol	*PawLog*	Imperative (C++)
Centralized	35	3229
Distributed	48	4445
Cross-layer	59	5817

Table 1: *PawLog* and Compiled C++ comparison

Compactness of *PawLog*. Table 1 illustrates the compactness of *PawLog*, by comparing the number of *PawLog* rules (2nd column) for three representative protocols against the actual number of lines of code (LOC) in the generated RapidNet and Gecode C++ code (3rd column).

Each *PawLog* program includes all rules required to implement routing and channel selection. These include rules for LSU propagation, route computation, channel selection, and dissemination of channel information in the network. The generated imperative code is approximately $100X$ the size of the equivalent *PawLog* program. The generated code is a good estimation on the LOC required by a programmer to implement these protocols in a traditional imperative language. In fact, *PawLog*'s reduction in code size should be viewed as a lower bound. This is because the generated C++ code implements only the rule processing logic, and does not include various PUMA's built-in libraries, e.g. Gecode's constraint solving modules, the network and channel abstraction layers provided by RapidNet. These built-in libraries need to be written only once, and are reused across all protocols written in *PawLog* rules.

While a detailed user study will allow us to comprehensively validate the usability of *PawLog*, we note that the orders of magnitude reduction in code size makes *PawLog* programs significantly easier to write, understand, debug and extend than multi-thousand-line imperative alternatives.

8. RELATED WORK

Several architectures and designs for dynamic spectrum access/sharing [20, 24, 19, 7] and channel selection and routing [10, 23, 6, 11, 25, 22] have been proposed for mitigating the impact of harmful interference and thus improving overall network performance. PUMA aims to to enable all of the above protocols and the policies that dictate their behaviors to be specified and customized easily.

XG [20] proposes an architecture for policy-based network management for spectrum access control. PUMA's policy-based framework similarly addresses the issue of channel management, but also provides the capabilities to implement policy-based routing protocols and perform cross-layer optimizations.

Prior work [13, 25] have formulated channel selection as COPs, however they are typically hard-coded into a constraint solver and limited to centralized contexts. [16] proposes the formulation of centralized channel selection policies as declarative COP programs. Our paper significantly extends this work by exploring centralized approximations, distributed channel selection, traffic-aware cross-layer optimizations, as well as an extensive experimental evaluation in both simulations and the ORBIT testbed.

Declarative networking has been studied in both wired [18] and wireless [15] environments, and even used as a basis for course projects in a networked systems class [12]. PUMA focuses on a new domain that combines declarative channel selection and routing in wireless mesh networks. By integrating a declarative networking engine with a constraint solver, PUMA provides novel capabilities that enable distributed cross-layer optimizations in an incremental fashion.

9. CONCLUSION

This paper presents PUMA, a policy-based extensible platform that combines channel selection and routing within a common declarative framework. PUMA integrates a declarative networking engine with a constraint solver to realize a variety of declarative wireless routing and channel selection protocols, and in addition, provides avenues to optimize across route and channel selection policies. We have developed a prototype of PUMA using the RapidNet declarative networking system and the Gecode constraint solver, and have carried out extensive evaluations of PUMA in simulation and on the ORBIT testbed. We are in the processing of releasing PUMA as open-source for use by the networking community. We are also exploring integrating PUMA with legacy systems, for instance, interfacing our channel selection manager with software router packages (e.g. Quagga [21]).

While our policy-based customizations demonstrate the flexibility and generality of PUMA, we further plan to explore runtime policy-based adaption, where even the optimization goals and constraints themselves can be reconfigured at runtime based on application requirements and network conditions. Furthermore, we plan to apply existing work on declarative network verification [27], in order to formally prove properties of *PawLog* protocols, e.g. reasoning about the interactions between channel selection and routing.

10. REFERENCES

[1] Gecode constraint development environment. http://www.gecode.org/.
[2] Meraki. http://meraki.com/.
[3] Network Simulator 3. http://www.nsnam.org/.
[4] ORBIT Wireless Network Testbed. http://www.orbit-lab.org/.
[5] RapidNet. http://netdb.cis.upenn.edu/rapidnet/.
[6] M. Alicherry, R. Bhatia, and L. E. Li. Joint channel assignment and routing for throughput optimization in multi-radio wireless mesh networks. In *MobiCom*, 2005.
[7] V. Brik, E. Rozner, S. Banerjee, and P. Bahl. DSAP: a protocol for coordinated spectrum access. In *DySPAN*, 2005.
[8] C. Chereddi, P. Kyasanur, and N. H. Vaidya. Design and implementation of a multi-channel multi-interface network. In *REALMAN*, 2006.
[9] T. Clausen and P. Jacquet. Optimized link state routing protocol (olsr). In *RFC 3626 (Experimental)*, October 2003.
[10] A. Dhananjay, H. Zhang, J. Li, and L. Subramanian. Practical, distributed channel assignment and routing in dual-radio mesh networks. In *SIGCOMM*, 2009.
[11] R. Draves, J. Padhye, and B. Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *MobiCom*, 2004.
[12] H. Gill, T. Saeed, Q. Fei, Z. Zhang, and B. T. Loo. An Open-source and Declarative Approach Towards Teaching Large-scale Networked Systems Programming. In *SIGCOMM Education Workshop*, 2011.
[13] K. Jain, J. Padhye, V. N. Padmanabhan, and L. Qiu. Impact of interference on multi-hop wireless network performance. In *MobiCom*, 2003.
[14] F. Kuhn and R. Wattenhofer. On the complexity of distributed graph coloring. In *PODC*, 2006.
[15] C. Liu, R. Correa, X. Li, P. Basu, B. Loo, and Y. Mao. Declarative policy-based adaptive MANET routing. In *ICNP*, 2009.
[16] C. Liu, X. Li, S. C. Muthukumar, H. Gill, T. Saeed, B. T. Loo, and P. Basu. A policy-based constraint-solving platform towards extensible wireless channel selection and routing. In *PRESTO*, 2010.
[17] M. Liu, W. Zhou, N. Taylor, Z. Ives, and B. T. Loo. Recursive Computation of Regions and Connectivity in Networks. In *ICDE*, 2009.
[18] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative Networking. In *Communications of the ACM (CACM)*, 2009.
[19] L. Ma, X. Han, and C.-C. Shen. Dynamic open spectrum sharing MAC protocol for wireless ad hoc networks. In *DySPAN*, 2005.
[20] F. Perich. Policy-based Network Management for NeXt Generation Spectrum Access Control. In *DySPAN*, 2007.
[21] Quagga Routing Suite. http://www.quagga.net/.
[22] Ramachandran, K. N. and Belding, E. M. and Almeroth, K. C. and Buddhikot, M. M. Interference-Aware Channel Assignment in Multi-Radio Wireless Mesh Networks. In *INFOCOM*, 2006.
[23] A. Raniwala, K. Gopalan, and T.-c. Chiueh. Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2004.
[24] C. Santivanez, R. Ramanathan, C. Partridge, R. Krishnan, M. Condell, and S. Polit. Opportunistic spectrum access: Challenges, architecture, protocols. In *ACM WiCon*, Boston, MA, 2006.
[25] A. Subramanian, H. Gupta, and S. Das. Minimum Interference Channel Assignment in Multi-Radio Wireless Mesh Networks. In *SECON*, 2007.
[26] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.
[27] A. Wang, P. Basu, B. T. Loo, and O. Sokolsky. Towards declarative network verification. In *11th International Symposium on Practical Aspects of Declarative Languages (PADL)*, 2009.
[28] Y. Yi and M. Chiang. Wireless Scheduling Algorithms with O(1) Overhead for M-Hop Interference Model. In *IEEE ICC*, 2008.

Appendix A: WCETT Metric

The *Weighted Cumulative Expected Transmission Time* (WCETT) [11] metric is based on *Expected Transmission Time (ETT)* of each link and takes into account path channel diversity.

$$WCETT_P = (1-\beta) * \sum_{l \in P} ETT_l + \beta * \max_{1 \le j \le k} X_j \quad (7)$$

In (7), the WCETT of a path P is expressed as a weighted formula (tunable by coefficient β) between the path ETT (summation of ETT ETT_l for all links in path P), and the maximum ETT of the bottleneck channel (computed by taking the max of all X_j, where X_j is defined as the sum of ETT for all links in P with channel assignment j).