University of Pennsylvania
ScholarlyCommons

Technical Reports (CIS) | Department of Computer & Information Science

January 2008

A Declarative Perspective on Adaptive MANET Routing

Changbin Liu
University of Pennsylvania

Yun Mao
University of Pennsylvania

Mihai Oprea
University of Pennsylvania

Prithwish Basu
BBN Technologies

Boon Thau Loo
University of Pennsylvania, boonloo@cis.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/cis_reports

Recommended Citation
Liu, Changbin; Mao, Yun; Oprea, Mihai; Basu, Prithwish; and Loo, Boon Thau, "A Declarative Perspective on Adaptive MANET Routing" (2008). *Technical Reports (CIS)*. Paper 871.
http://repository.upenn.edu/cis_reports/871

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-08-03.

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_reports/871
For more information, please contact repository@pobox.upenn.edu.

A Declarative Perspective on Adaptive MANET Routing

Abstract
In this paper, we present a declarative perspective on adaptable extensible MANET protocols. Our work builds upon declarative networking, a recent innovation for building extensible network architectures using declarative languages. We make the following contributions. First, we demonstrate that traditional MANET protocols, ranging from proactive, reactive, to epidemic can be expressed in a compact fashion as declarative networks, and we validate experimentally the use of declarative techniques to implement traditional MANETs emulated on a testbed cluster. Second, we show that the declarative framework enables policy-driven adaptation, in which a generic set of declarative rule-based policies are used to make runtime decisions on the choice of MANET protocols. Third, we present some initial ideas on fine-grained protocol composition and adaptation, where a typical MANET protocol can be composed and adapted from simpler components.

Comments
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-08-03.

This technical report is available at ScholarlyCommons: http://repository.upenn.edu/cis_reports/871

A Declarative Perspective on Adaptive MANET Routing

Changbin Liu* Yun Mao* Mihai Oprea* Prithwish Basu† Boon Thau Loo*
*University of Pennsylvania †BBN Technologies

ABSTRACT

In this paper, we present a declarative perspective on adaptable extensible MANET protocols. Our work builds upon declarative networking, a recent innovation for building extensible network architectures using declarative languages. We make the following contributions. First, we demonstrate that traditional MANET protocols, ranging from proactive, reactive, to epidemic can be expressed in a compact fashion as declarative networks, and we validate experimentally the use of declarative techniques to implement traditional MANETs emulated on a testbed cluster. Second, we show that the declarative framework enables policy-driven adaptation, in which a generic set of declarative rule-based policies are used to make runtime decisions on the choice of MANET protocols. Third, we present some initial ideas on fine-grained protocol composition and adaptation, where a typical MANET protocol can be composed and adapted from simpler components.

1. INTRODUCTION

In the past decade there has been intense activity on the development of routing protocols for mobile ad hoc networks (MANET). The unpredictable wireless channel and mobility of nodes cause MANETs to be extremely dynamic in nature and hence very different from the wired networks that constitute the public Internet. Due to a wide range of variability in network connectivity and also a wide range of data traffic patterns, a *one-size-fits-all* MANET routing algorithm does not exist. Hence a wide variety of routing protocols have been proposed in the past few years, all with their own strengths and weaknesses, and all with varying degrees of success. For example, reactive routing protocols such as DSR [6] and AODV [11] set up routing state *on demand* and hence are ideal for low traffic environments; proactive routing protocols such as OLSR [2] on the other hand expend network bandwidth to gather routing state with a purpose of amortizing this extra cost over multiple traffic flows – hence these are better for high traffic load environments, in general. Recently researchers have focused on the disruption tolerance aspects of MANETs that are at best intermittently connected, e.g., epidemic routing protocols.

Hybrid routing protocols attempt to address the above problem by combining features from various *pure* protocols of types such as proactive, reactive, and epidemic. While extant protocols in the hybrid category [4, 8] have systematic logic behind their design, they are too restrictive and are specified in a *stove-piped* manner. Ideally, we would like to create generic hybrid protocols by composition of any number of known protocols provided the policies, rules, and conditions for switching amongst them are clearly specified. Recently, there has been some interest in policy-based decision making at run time in wireless networks, most notably in the areas of dynamic spectrum access and security; however, there has been little impact of these on the design of MANET routing protocols.

Another recent research initiative that attempts to compose complex routing protocols from simpler components at a finer granularity than hybrid protocols is *Component Based Routing* [5]. While their goal is adaptability to the dynamic environment in MANETs, the focus is on the diagnosis and the subsequent improvement of a weak protocol component.

In this paper, we present our initial exploration into using declarative languages for specification of MANET protocols and the composition rules for creating hybrid protocols that accommodate policies as first class concerns. The key advantage is that the policies and network specifications are in the same declarative framework, suggesting opportunities for cross-layer optimizations in future.

Our work builds upon declarative networking [9], and makes the following three contributions. First, we demonstrate that MANET protocols such as DSR, Link State routing, and Epidemic can be specified tersely using the *NDlog* declarative networking [9] language. We validate these protocols by executing them on MANETs (up to 80 nodes) emulated on a testbed cluster. Second, we demonstrate how policy-based decisions for creating hybrid protocols can be expressed in the same declarative language, and used to switch between protocols. Third, we present some initial ideas on fine-grained protocol composition by using component-based abstractions where functionalities are exposed as logical predicates in the declarative language. These ideas serve as a building block towards our grand vision of building adaptive MANET routing protocols from simpler components.

2. BACKGROUND

We first begin with a brief overview of declarative networking. The high level goal of *declarative networks* is to build extensible architectures that achieve a good balance of flexibility, performance and safety. Declarative networks are specified using *Network Datalog* (*NDlog*), which is a distributed recursive query language used for querying network graphs.

Declarative queries such as *NDlog* are a natural and compact way to implement a variety of routing protocols and overlay networks. For example, traditional routing protocols such as the path vector and distance-vector protocols can be expressed in a few lines of code [9], and the Chord distributed hash table in 47 lines of code. When compiled and executed, these declarative networks perform efficiently relative to imperative implementations.

NDlog is based on Datalog [12]: a Datalog program consists of a set of declarative *rules*. Each rule has the form `p :- q1, q2, ..., qn.`, which can be read informally as "`q1` and `q2` and ... and `qn` implies `p`". Here, `p` is the *head* of the rule, and `q1`, `q2`,...,`qn` is a list of *literals* that constitutes the *body* of the rule. Literals are either *predicates* with *attributes* (which are bound to variables or constants by the query), or boolean expressions that involve function symbols (including arithmetic) applied to attributes.

Datalog rules can refer to one another in a mutually recursive fashion. The order in which the rules are presented in a program is semantically immaterial; likewise, the order predicates appear in a rule is not semantically meaningful. Commas are interpreted as logical conjunctions (*AND*). The

names of predicates, function symbols, and constants begin with a lowercase letter, while variable names begin with an uppercase letter. Function calls are additionally prepended by `f_`. Aggregate constructs are represented as functions with attribute variables within angle brackets (<>). We illustrate *NDlog* using a simple example of two rules that computes all pairs of reachable nodes:

```
r1 reachable(@S,N) :- link(@S,N).
r2 reachable(@S,D) :- link(@S,N), reachable(@N,D).
```

The rules `r1` and `r2` specify a distributed transitive closure computation, where rule `r1` computes all pairs of nodes reachable within a single hop from all input links (denoted by the `neighbor`, and rule `r2` expresses that "if there is a link from `S` to `N`, and `N` can reach `D`, then `S` can reach `D`." By modifying this simple example, we can construct more complex routing protocols, such as the distance vector and path vector routing protocols.

NDlog supports a *location specifier* in each predicate, expressed with the `@` symbol followed by an attribute. This attribute is used to denote the source location of each corresponding tuple. For example, all `reachable` and `link` tuples are stored based on the `@S` address field. The output of interest is the set of all `reachable(@S,D)` tuples, representing reachable pairs of nodes from `S` to `D`.

2.1 Dataflow Execution and Tables

NDlog queries are compiled and executed as *distributed dataflows* by the query processor to implement various network protocols. These dataflows share a similar execution model with the Click modular router [7], which consists of elements that are connected together to implement a variety of network and flow control components. In addition, elements include database operators (such as joins, aggregation, selections, and projects) that are directly generated from the *NDlog* rules. Messages flow among dataflows executed at different nodes, resulting in updates to local tables, or query results that are returned to the mobile hosts that issued the queries. The local tables store the state of intermediate and computed query results, which include the network state of various network protocols.

Predicates refer to tables which themselves are declared as soft-state with lifetimes. Event predicates (denoted with an additional "e" in this paper) are used to denote transient tables which are used as input to rules but not stored. For example, utilizing P2's [1] built-in `periodic` keyword , node `X` periodically generates a `ePing` event every 10 seconds to its neighbor `Y` denoted in the `link(@X,Y)` predicate:

```
ePing(@Y,X) :- periodic(@X,10), link(@X,Y).
```

2.2 Modularity and Composability

In order to support network functionality composition and code reuse, we recently proposed language extensions to *NDlog* provide support for *Composable Virtual Views* (CViews) [10], which define rule groups that, when executed together, perform a specific functionality. The syntax of CViews is as follows:

```
viewName(K1,K2,...,Kn, &R1,&R2,...,&Rm)
```

Each CView predicate has an initial set of attributes `K1,K2, ...Kn` which are already bound to input values read from another predicate (intuitively, these are like input parameters to a function call). The remaining attributes, `&R1,&R2,...,&Rm`, represent the *return values* from invoking the predicate given the input values. We illustrate using a view definition for the following CView predicate `ePing(@SrcAddr, DestAddr, &RTT)`:

```
def ePing(@Src,Dest,&RTT) {
  p1 this.eReq(@Dest,Src,T) :-
    this.init(@Src,Dest), T=f_now().
  p2 this.eResp(@Src,T) :- this.eReq(@Dest,Src,T).
  p3 this.return(RTT) :-
    this.eResp(@Src,T), RTT=f_now()-T. }
```

Any rule that must compute the RTT between two nodes can simply include the `ePing` event predicate in the rule body. `this` is a keyword used to express the context of the CView. All predicates beginning with `this` are valid only locally within the `ePing` CView. There are two new built-in events/actions: `this.init` and `this.return`. Rule `p1`, upon receiving event `this.init` along with the query keys `Src` and `Dest`, takes the current timestamp `T`, and passes the data to the host `Dest` as a ping request. After the destination node receives it in rule `p2`, a ping response event is immediately sent back to the source with the timestamp. In rule `p3`, the source node computes the round trip time based on the timestamp and issues a `this.return` action that finishes the query processing.

3. DECLARATIVE ROUTING IN MANETS

In this section, we demonstrate how a variety of MANET protocols, ranging from proactive, reactive, to epidemic protocols can be expressed using the declarative framework. This section sets the stage for Section 4 where we discuss policies for hybridizing and switching between different protocols.

3.1 Proactive Protocol: Link State

A well-known proactive MANET protocol is OLSR (Optimized Link State Protocol) [2]. We show an example for network-wide flooding of link-state (LS) updates, as expressed by the following *NDlog* rules:

```
ls1 lsu(@S,S,N,C,S) :- link(@S,N,C)
ls2 lsu(@M,S,N,C,Z) :- link(@Z,M,C1),
          lsu(@Z,S,N,C,W), M!=W.
```

`lsu(@M,S,N,C,Z)` is a link state update (LSU) corresponding to `link(S,N,C)`. This tuple is flooded in the network starting from source node `S`. During the flooding process, node `M` is the current node it is flooded to, while node `Z` is the node that forwarded this tuple to node `M`.

Rule `ls1` generates a `lsu` tuple for every link at each node. Rule `ls2` states that each node `Z` that receives a `lsu` tuple recursively forwards the tuple to all neighbors `M` except the node `W` that it received the tuple from. Datalog tables are set-valued, meaning that duplicate tuples are not considered for computation twice. This ensures that no similar `lsu` tuple is forwarded twice.

To flood the LSUs periodically, one can utilize the `periodic` keyword, to flood once in every period. Also, if efficient flooding is desired (as in OLSR), the following rules `olsr1-2` are modified from the previous two rules to only forward LSUs to a subset of neighbors known as multipoint relays (MPR)[1], denoted as `mpr` predicates (M is an MPR of Z) which themselves could be defined with additional rules, but are omitted here due to paucity of space.

[1]The union of the neighbor sets of MPRs of any node X is equal to the set of 2-hop neighbors of X.

```
olsr1 lsu(@S,S,N,C,S) :- periodic(@S,10),
        link(@S,N,C).
olsr2 lsu(@M,S,N,C,Z) :- mpr(@Z,M,C1),
        lsu(@Z,S,N,C,W), M!=W.
```

Once the entire network topology, i.e., all the links, are available at each node, additional rules are required in order to compute the shortest paths with minimum cost `C` for each source `S` and destination `D`. These rules take as input the local `lsu` tuples generated, and essentially result in the execution of the Dijkstra's algorithm locally. They are shown as follows:

```
bp1 path(@M,S,N,P,C) :- lsu(@M,S,N,C,W),
          P=f_init(S,N).
bp2 path(@M,S,D,P,C) :- lsu(@M,S,N,C1,W),
          path(@M,N,D,P2,C2),
          C=C1+C2, P=f_concatPath(S,P2).
bp3 bestPathCost(@M,S,D,min<C>) :- path(@M,S,D,P,C).
bp4 bestPath(@M,S,D,P,C) :- bestPathCost(@M,S,D,C),
          path(@M,S,D,P,C).
```

In rule `bp1`, paths with two hops are built from every link, while in rule `bp2` paths are recursively constructed by concatenating shorter path with links. Rule `bp3` computes the minimum cost for paths with same sources and destinations, and rule `bp4` finally computes the best path.

3.2 Reactive Protocol: Source Routing

Next, we demonstrate a reactive protocol based on DSR. The following set of rules show the route discovery of DSR (rules `dsr1-4`) followed by the route response (rules `dsr5-6`) traversing the best reverse path from destination to source.

```
dsr1 eRouteReq(@N,S,D,P,C) :- eQuery(@S,D),
          link(@S,N,C), P=f_init(S).
dsr2 eRouteReq(@Z,S,D,P,C):-
          shortestRoute(@N,S,D,P1,C1),
          link(@N,Z,C2),
          C=C1+C2, P=f_concatPath(P1,S).
dsr3 minCost(@N,S,D,min<C>) :-
          routeReq(@N,S,D,P,C).
dsr4 shortestRoute(@N,S,D,P,C) :-
          minCost(@N,S,D,C),
          routeReq(@N,S,D,P,C).
dsr5 eRouteReply(@Z,S,D,P2,P1,C) :-
          eRouteReq(@N,S,D,P2,C), N==D,
          Z=f_last(P2),P1=f_removeLast(P2).
dsr6 eRouteReply(@Z,S,D,P,P1,C) :-
          eRouteReply(@Z,S,D,P,P2,C), Z=f_last(P2),
          f_size(P2)>0, P1=f_removeLast(P2),
```

In DSR, a requesting node `S` issues an initial route request, denoted by `eQuery(@S,D)` event in rule `dsr1`. This results in a `eRouteReq` message tuples that is generated and recursively forwarded along all links (rules dsr2). The `routeReq` table is used to cache current route requests. To prune unnecessary paths, rules `dsr3-4` ensures that only the shortest path from the initial node `S` to the intermediate node `N` is maintained.

Upon reaching the destination node `D`, rule `dsr5` generates a `eRouteReply` message that is then sent back recursively via rule `dsr6` along the computed best reverse path back to the requesting node `S`. The functions `f_last` and `f_removeLast` returns and removes the last node from a path respectively. Rule `dsr6` reaches the initial requesting node `S` when the remaining path is of length 0.

The rules for AODV [11] share similarities with DSR above, where only the next hop rather than the entire path is maintained.

3.3 Epidemic Protocols

Epidemic routing [16] and its variants such as PREP [13] have been proposed for reliable delivery in intermittently connected MANETs[2]. A key reliability component of such protocols is the summary vector exchange as illustrated by the rules `e1-4` below:

```
e1 eBitVecReq(@Y,X,V):- summaryVec(@X,V),
          eDetectNewLink(@X,Y).
e2 eBitVecReply(@X,Y,V):- eBitVecReq(@Y,X,V1),
          summaryVec(@Y,V2),
          V=f_vec_AND(V1,f_vec_NOT(V2)).
e3 eNewMsg(@Y,I,S,D):- eBitVecReply(@X,Y,V),
          msgs(@X,I,S,D),
          f_vec_in(V,I)==true.
e4 msgs(@Y,I,S,D):- eNewMsg(@Y,I,S,D).
```

In rule `e1`, node `X` detects that a new link comes to be available, then it retrieves its local (`summaryVec`) table, consisting a bit vector where the ith bit denotes the receipt of the ith message, and then generates a `eBitVecReq` request to the neighbor `Y` connected by the new link. Upon receiving the request, node `Y` performs a bitwise AND operation between the incoming summary vector `V1` and the negation of local summary vector `V2` to generate a new vector `V` which is sent back to `X`. This new vector `V` denotes messages seen by `X` but not `Y`. Rules `e3-4` then enables node `X` to filter local messages to be sent based on the bit vector `V` stored in the reply, which are then buffered in the local `msgs` table for transmission.

PREP is an extension of the basic epidemic protocol, where transmissions of messages are prioritized based on the remaining lifetimes of individual messages. This can be done elegantly in *NDlog* by sorting the `msg` table based on a user-defined ranking function, and then have forwarding rules that take the `msgs` table as input, and transmit based on the sorted order.

3.4 Preliminary Evaluation

We perform an initial evaluation of the three classes of MANET protocols described above: proactive LS (Section 3.1), reactive DSR (Section 3.2), and epidemic (Section 3.3). We base our evaluation on the P2 declarative networking system [1]. We focus on evaluating the aggregate bandwidth utilization for all protocols, and in the case of the epidemic protocol, we additionally measured the message delivery time given a disconnected network. The experiment for our evaluation is carried out on a local cluster with eight Pentium IV 2.8GHz PCs with 2GB RAM running Fedora Core 6 with kernel version 2.6.20, which are interconnected by high-speed Gigabit Ethernet. When executing the LS and DSR protocols, we emulate connectivity in a wireless environment by initializing the `link` table of each node such that every node has three neighbors. This setup allows us to validate the scalability trends (in terms of bandwidth utilization)) of each protocol. As future work, we intend to experiment with executing our system on a wireless testbed such as ORBIT [15]. On an actual testbed, the rules to be executed remain the same, with the main difference being that the `link` table at each node will be determined by actual wireless connectivity. We summarize our results as follows:

Link-state protocol: In our first experiment, we execute the LS protocol as described in Section 3.1. In our protocol, we measure the aggregate communication overhead re-

[2]These are a class of disruption tolerant networks or DTNs.

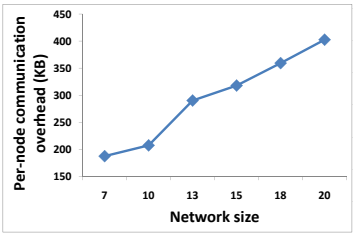

Figure 1: Per-node Communication Overhead (KB) for LS

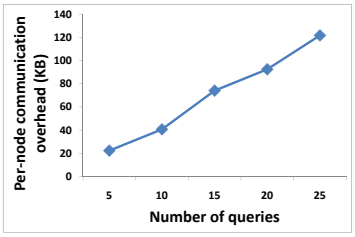

Figure 2: Per-node Communication Overhead (KB) for DSR.

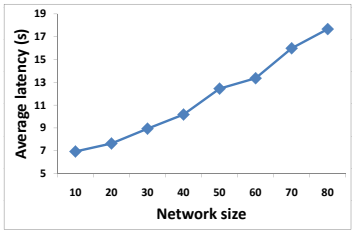

Figure 3: Average Message Delivery Latency (s) for Epidemic

quired for each node to propagate all its LSUs to all other nodes. Upon receiving the LSUs, each node then executes rules for computing the shortest paths. Figure 1 shows that the per-node communication overhead increases linearly as the network size increases, a scalability trend that one would expect in a link state protocol.

Dynamic source routing: In our second experiment, we execute the DSR protocol as described in Section 3.2. Given a network size of 20, we generated route request queries from random sources to destinations. Figure 2 shows that, as expected, the per-node communication overhead increases linearly as the number of route request increases. We note a similar linear trend when we fix the number of queries, but increase the network size.

Epidemic: In our final experiment, we measured the average message delivery latency of the epidemic protocol as described in Section 3.3. In our experimental setup, we emulated a partially connected network by ensuring that at any point in time, only 10% of all possible links are available, and periodically, changing the availability of links. Given the constant churn of link availability, messages are injected into network from random sources and epidemically routed until reaching their destinations. For each message, we measured the *delivery latency*, which is determined by the time lag between message injection and arrival at the destination. Figure 3 shows the average message delivery latency for a setup where five messages are injected every second into the network. The average latency increases approximately linearly as the network size increases. As we increase the rate at which messages are injected, the graph becomes quadratic as queuing delays increase the time required for messages to be routed to their destinations.

4. ADAPTIVE ROUTING IN MANETS

Building upon the basic declarative MANET protocols in the previous section, we demonstrate the construction of *composite* protocol behaviors by applying appropriate policies to available declarative protocol code. Such protocol behaviors are often necessary in MANETs for adapting to a wide range of network and traffic conditions. We describe two mechanisms for achieving this: *policy-driven hybrid protocols* and *component-based routing*.

4.1 Policy-Driven Hybrid Protocols

Hybrid protocols attempt to combine the best features from various *pure* protocols and attempt to operate in one of the constituent modes depending on the network dynamics, traffic conditions, and other requirements such as reliability, security etc. If there were a perfect oracle that could learn about the entire network state, it would be easy to write switching rules for representing hybrid protocols in the sense that a node could decide to always run the optimal protocol that is most appropriate for the current scenario.

In the absence of this global information at every node in the network, it is still possible to write a generic set of policies that can allow run-time adaptation based on local state available at each node. However, characterizing the optimality and stability of such composite protocols in a general sense is an open and orthogonal problem. In this paper, we focus on demonstrating how hybrid composition could be facilitated with little effort using the declarative paradigm. Stability and optimality analysis is an interesting topic for future research.

4.1.1 Hybrid Link State Routing

A well-known LS routing variant for handling moderate to high rate of change in network topology is Hazy Sighted Link State routing (HSLS) [14]. This protocol attempts to control the scope and frequency of its LSU broadcast scheme based on the topology of the network. The basic principle of HSLS is that a node should not be affected significantly by link dynamics due to mobility or failure in a portion of network that is far away from it. Hence unlike the pure LS protocol which performs a network wide flood of all LSUs, HSLS sends LSUs to the k hop neighbors of a node with a period equal to $2^{k-1}T$, where T is a nominal period. If link dynamics are high, pure LS starts thrashing because remote nodes could receive an LSU corresponding to a link that has long vanished.

Policy rules used in HSLS can easily be expressed as follows:

```
hsls1 lsu(@S,S,N,C,S,K) :- periodic(@S,T),
          link(@S,N,C), T=f_pow(2,K-1),
          K=range[1,10].
hsls2 lsu(@M,S,N,C,Z,K-1) :- lsu(@Z,S,N,C,W,K),
          link(@Z,M,C1), K>0, M!=W.
```

The primary disadvantages of HSLS is that it sacrifices optimality in routing to the need for scalability. This is because it gathers imperfect information about the network topology and computes routes on this topology. Imperfect topology knowledge may result in computation of suboptimal routes, and this effect can be pronounced in somewhat dynamic but sparsely connected topologies. Hence, if the link *average availabilities* (AA) [13] (a metric that is described in detail in Section 4.2) stop fluctuating wildly in most parts of the network as indicated by gathered LSUs, one may decide to

revert back to pure LS routing since that may yield near-optimal routes with a lower *stretch*. Note that rule `hsls1` is periodically fired, and the period of execution depends on 2^{K-1}.

Based on the HSLS rules above, one can further define a generic policy that allow us to switch between HSLS and LS based on the computed average AA of all links collected in the network. The average AA threshold below which to switch to pure LS is a configuration parameter that is set either by analysis or experimentation.

```
#include ls1, ls2, hsls1, hsls2
#define THRES 0.5
s1 averageLinkAvail(@M,AVG<AA>) :-
          lsu(@M,S,N,AA,Z,K).
s2 useHSLS(@M) :- averageLinkAvail(@M,AA), AA>THRES.
s3 useLS(@M) :- averageLinkAvail(@M,AA), AA<=THRES.
```

`#include` is a macro used to include earlier rules. Rule `s1` computes at node M, the average AA of all links gathered from the different LSUs that pass through M. Rules `s2-3` generate `useHSLS` and `useLS` predicates which are then added to rules `hsls1` and `ls1`, respectively.

To encode a new policy (e.g. use LS instead of HSLS when the network is sparse with high frequency of link updates), one only needs to modify the above rules to generate `useHSLS` and `useLS` without having to change the rules for the individual protocols themselves. The main point is not whether one policy is superior to another, but rather that the declarative framework makes specification of such policies concise and flexible.

4.1.2 Hybrid Proactive-Epidemic

As an alternative example, one can utilize a hybrid proactive-epidemic protocol, useful in a disruption-tolerant setting. This hybrid protocol switches between two modes of operation: (1) single path LS routing in well connected parts of the network, and (2) multi-path epidemic style routing in disrupted parts of the network. This is similar to the Anxiety-Prone Link-State (APLS) protocol [8] developed for the SPINDLE DTN project. The following set of rules demonstrate the ease at which these policies are implemented:

```
#include bp1, bp2, bp3, bp4
#define THRES 1.2
pe1 useEpidemic(@M,S,D) :- bestPath(@M,S,D,P,C),
          C>THRES.
pe2 useLS(@M,S,D) :- bestPath(@M,S,D,P,C), C<=THRES.
```

In the rules above, at any node M, `bestPath` computes the cost of the shortest path between S and D, which is then used by rules `pe1-2` to determine whether to use LS or epidemic routing. Intuitively, low values of path cost indicate that S and D are in a connected component whereas high values indicate that link availability is low, or LS information is unavailable or stale, hence an epidemic protocol is desired under those circumstances to improve delivery probability.

Note that while the first example illustrates switching the underlying dissemination scheme, the latter illustrates how to switch the route computation and forwarding. The declarative framework and compact specifications make it easy to write other (more intelligent) switching rules in this scenario.

4.2 Component Based Routing

In the second aspect of adaptive MANETs, we describe how components of certain routing protocols, if specified declaratively, could become useful in protocol composition. We utilize the CView functionality described in Section 2.2 for composing rules into modular groups. Using CViews, a set of basic components that are used by proactive, reactive, or epidemic routing protocols can become declarative building blocks and could be shared across a multitude of adaptive protocols. The components could be executed upon occurrence of certain events (like in the event-condition-action paradigm). The vision is that a routing protocol can evolve depending on network and traffic conditions rather than being *stove-piped* as they currently are. Examples of such components include neighbor discovery by periodic heartbeats, path discovery by scoped flooding, path computation and maintenance on local topology database, and epidemic synchronization of data among neighbors using summary vectors. We illustrate two such components below. A protocol designer can use such components in bodies of rules similar to the ones shown in Section 4.1 and develop new adaptive MANET routing protocols.

4.2.1 Link Availability Component

Short for *average availabilities*, AA [13] is a novel metric of link used to measure the average fraction of time in the near future that the link will be available for use. AA is calculated as the total time since a link was detected divided by the total time that link's status is up. AA can be encapsulated as a neighbor discovery module, which may be the most straightforward sharable component reusable across multiple protocols. We present a more complex `linkAA` CView example as following rules for computing link availability used in our earlier discussion of hybrid protocols.

```
def linkAA(@S,N,&AA,&Status) {
  aa1 this.return(AA,Status):-
    this.init(@S,N), link(@S,N,UP,T_b,T_p,T_up),
    AA=(T_up+f_now()-T_p)/(f_now()-T_b), Status=UP.
  aa2 this.return(AA,Status):-
    this.init(@S,N), link(@S,N,DOWN,T_b,T_p,T_up),
    AA=T_up/(f_now()-T_b), Status=DOWN.
  aa3 link(@S,N,UP,f_now(),f_now(),0):-
    eDetectNewLink(@S,N).
  aa4 link(@S,N,UP,T_b,f_now(),T_up):-
    eDetectOldLink(@S,N),
    link(@S,N,DOWN,T_b,T_p,T_up).
  aa5 link(@S,N,DOWN,T_b,f_now(),T_up1):-
    eDetectFailLink(@S,N),
    link(@S,N,UP,T_b,T_p,T_up),
    T_up1=T_up +(f_now()-T_p). }
```

For each node `S`, the `linkAA` CView takes as input the neighbor `N`, and returns the `AA` metric and status `Status` for the link from `S` to `N`. The `link` table contains six attributes to store link information: (1) where this link is stored, (2) the neighbor endpoint of the link, (3) link's status which can be up or down denoted by `UP` and `DOWN`, (4) `T_b` which means the time when this link was originally detected, (5) `T_p` which means the most recent time when link's status changed, and (6) `T_up` which means the total link uptime.

When the `linkAA` CView is used as input in a rule body, if link's status is up, its total uptime will be T_up plus current time minus `T_p`, thus in rule `aa1` AA will be calculated as the total time since this link was detected divided by the total uptime. If link's status is down, its total uptime will be `T_up`, and this situation is stated in rule `aa2`. In rule `aa3`, when the node detects a new link which has never been seen before, it will be put into link table by setting `T_b` and `T_p` to be current time and `T_up` to be zero. In rule `aa4`, if the

node detects an old link which was down but comes to be available again, it only needs to set `T_p` to be current time. In rule `aa5`, if the node detects a link which becomes down, it updates `T_up` and sets `T_p` to be current time.

4.2.2 Parameterized Flood Component

Flooding is a necessary component that is useful in most MANET routing protocols, although each protocol performs flooding slightly differently. For example, pure LS routing floods LSUs to all nodes; OLSR floods LSUs via MPRs; HSLS alters the rate and scope of flooded LSUs; Gossip protocols add a probabilistic aspect to flooding to trade off coverage and overhead; Epidemic algorithms add a reliability component by means of summary vectors.

Despite these different uses, the important aspects that characterize a flooding scheme in the context of routing are: *what* is being flooded, *which* nodes are participating, *how* far a flooded packet goes and *when* it is initiated. We present a CView for flooding that captures the above concerns as arguments such that the component is usable by several routing protocols with different instantiations. This requires an extension to the P2 system so that table names can be passed as argument parameters to the CView.

```
def flood(@S,TTL,Payload,Nbr,Sched) {
  f1 floodMsg(@S,S,TTL,Payload,Nbr,Sched,T) :-
       this.init(@S,S,TTL,Payload,Nbr,Sched,T),
       Sched(@S,TTL,0,T).
  f2 floodMsg(@N,S,TTL-1,Payload,Nbr,Sched,T) :-
       floodMsg(@M,S,TTL,Payload,Nbr,Sched,T1),
       Nbr(@M,N,C1), TTL>0, Sched(@S,TTL,T1,T). }
```

The `flood` CView is initiated at node `S`, taking as input payload `Payload`, scope `TTL`, and table names represented by variables `Nbr` and `Sched`. `Nbr` determines which neighbors should receive the flooded message, and `Sched` determines the conditions when the flood happens. Rule `f1` initiates the flooding at node `S` given the input parameters of the CView. Executing rule `f2` will recursively flood the `Payload` up to `TTL` hops in a subgraph sketched out by the `Nbr` predicate in the manner dictated by `Sched`. Examples of `Nbr` include `link` (if we want to flood everywhere) and `mpr` (if we only want to flood via MPRs); `Sched` may be set such that floods are fired upon link UP events, or periodically by HSLS rules etc. Note that `Payload` can be regular data or neighbor table fragments. Optimizations proposed in OLSR [2] such as flooding only the MPR links are possible by passing the appropriate tuples here.

The `flood` CView can be used by DSR for flooding route requests, and by LS for flooding LSUs. Interestingly, one can also write a CView on epidemic forwarding using summary vectors (basically encapsulate the rules `e1-4` in Section 3.3 into a CView), which can be used by both DSR and LS for propagating route requests and LSUs respectively in a more disconnected setting. The choice of using the epidemic or traditional flood CView can then be determined using policy rules. Alternatively, one can call the summary vector CView from inside the flood CView to get the desired behavior. We have not shown those rules here due to space constraints.

5. CONCLUSION

In this paper, we present a declarative perspective on adaptive MANET routing. We demonstrate that a variety of MANET protocols can be specified tersely using the *ND-log* declarative networking language. Moreover, policy-based decisions can be expressed within the declarative framework for runtime switching amongst protocols. We further present initial ideas on fine-grained protocol composition using component based abstractions where functionalities are exposed as logical predicates in the declarative language.

Our immediate steps include experimentation on existing wireless testbeds [15] as well as emulation software [8] that enables us to evaluate our system in emulated wireless and DTN settings. Beyond experimentation, we plan to investigate techniques for analyzing stability and correctness of the composed protocols under a different network dynamics and traffic patterns. We believe that our declarative framework is ideal for exploring this complex space. We plan to further investigate integrating our techniques with recent attempts at building verifiable network specifications [3], especially in the context of stability and optimality analysis of hybrid MANETS. We also plan to explore cross-layer decision making in cognitive radio architectures.

6. REFERENCES

[1] P2: Declarative Networking. `http://p2.cs.berkeley.edu`.

[2] Clausen, T., and Jacquet, P. Optimized link state routing protocol (olsr). In *RFC 3626 (Experimental)* (October 2003).

[3] Griffin, T. G., and Sobrinho, J. L. Metarouting. In *ACM SIGCOMM* (2005).

[4] Haas, Z. J. A New Routing Protocol for the Reconfigurable Wireless Networks. In *IEEE Int. Conf. on Universal Personal Communications* (1997).

[5] Huang, H., and Baras, J. S. Component based routing: A new methodology for designing routing protocols for manet. In *25th Army Science Conference* (2006).

[6] Johnson, D. B., and Maltz, D. A. Dynamic Source Routing in Ad Hoc Wireless Networks. In *Mobile Computing*, vol. 353. 1996.

[7] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. The Click Modular Router. *ACM Transactions on Computer Systems 18(3)* (2000), 263–297.

[8] Krishnan, R., Basu, P., Mikkelson, J. M., Small, C., Ramanathan, R., Brown, D., Burgess, J., Caro, A., Condell, M., Goffee, N., Hain, R. R., Hansen, R., Jones, C., Kawadia, V., Mankins, D., Schwartz, B., Strayer, T., Ward, J., Wiggins, D., and Polit, S. The spindle disruption-tolerant networking system. In *MILCOM* (2007).

[9] Loo, B. T., Hellerstein, J. M., Stoica, I., and Ramakrishnan, R. Declarative Routing: Extensible Routing with Declarative Queries. In *ACM SIGMOD* (2005).

[10] Mao, Y., Loo, B. T., Ives, Z., and Smith, J. M. MOSAIC: Multiple Overlay Selection and Intelligent Composition. University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-07-22, 2007.

[11] Perkins, C. E., and Royer, E. M. Ad hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications* (New Orleans, LA, February 1999).

[12] Ramakrishnan, R., and Ullman, J. D. A Survey of Research on Deductive Database Systems. *Journal of Logic Programming 23*, 2 (1993), 125–149.

[13] Ramanathan, R., Hansen, R., Basu, P., Rosales-Hain, R., and Krishnan, R. Prioritized epidemic routing for opportunistic networks. In *ACM MobiOpp '07* (San Juan, Puerto Rico, 2007), pp. 62–66.

[14] Santivanez, C., Ramanathan, R., and Stavrakakis, I. Making link-state routing scale for ad hoc networks. In *ACM MobiHoc '01* (Long Beach, CA, 2001).

[15] Testbed, O. W. N. `http://www.winlab.rutgers.edu/docs/focus/ORBIT.html`.

[16] Vahdat, A., and Becker, D. Epidemic routing for partially-connected ad hoc networks. Tech. Rep. CS-200006, Duke University, April 2000.