

MANETkit: A Framework for MANET Routing Protocols

RAJIV RAMDHANY AND GEOFF COULSON

*Computing Department Lancaster University, South Drive, Lancaster, LA1 4WA, UK.
E-mail: {r.ramdhany, geoff}@comp.lancs.ac.uk*

Received: June 22, 2008. Accepted: December 11, 2009

Research in MANETs has resulted in the development of numerous and diverse routing protocols. We argue in this paper that this diversity is inherent to the MANET domain and therefore it will be important in future environments to simultaneously support multiple MANET protocols. On this basis, we propose a highly configurable component framework that facilitates the support of multiple MANET protocols and accommodates pluggable protocol functionality. Importantly, the framework also allows protocols to be composed, decomposed and hybridised in a variety of ways to create value-added functionality. In addition, it has coherent support for dynamic reconfiguration which opens the possibility for protocol hybridisation strategies to be safely executed at run-time. The paper outlines the functionality of the framework, illustrates its configurability, and offers a preliminary performance evaluation that demonstrates acceptable overhead.

Keywords: MANET, Ad hoc routing protocols, protocol framework, middleware, protocol configuration and hybridisation

1 INTRODUCTION

Mobile ad hoc networks (MANETs) require routing protocols to ensure that out-of-range nodes can communicate with each other via intermediate nodes. This task is by no means simple as MANETs vary radically (and dynamically) in size, density and traffic patterns. As a consequence, MANET researchers have proposed a plethora of routing protocols - e.g. AODV [1], DYMO [2], OLSR [3], ZRP [4], TORA [5], CGSR [6] to name a few—that are based on

varying design philosophies and designed to meet specific requirements from various application domains.

We argue in this paper that this diversity is inherent to the MANET domain and therefore it will be important in future environments to support multiple MANET protocols (including simultaneously). On this basis, we propose a highly configurable component framework that facilitates the support of multiple MANET protocols and accommodates pluggable protocol functionality. This goes beyond other frameworks developed by the MANET community—e.g., ASL [7] and PICA [8], and more general modular router frameworks like Click [9]—in being much broader in scope (see Section 8).

The design of our framework, which we call Manetkit, builds on our past experience of developing frameworks for middleware systems [10, 11] and overlay protocols [12]. Its aims are to i) reduce MANET implementation effort, ii) enhance the portability of protocol implementations, iii) facilitate the exploration of protocol optimisation/hybridisation efforts, and iv) to seamlessly integrate MANET routing in a wider middleware framework. A final aim, which we are now beginning to explore, is to support dynamic reconfiguration in MANET protocols.

The remainder of this paper is organised as follows: Section 2 describes the motivating ideas driving our work. Section 3 then provides background on the architectural abstractions we use in our design and Section 4 presents our framework’s architecture. Sections 5 and 6 evaluate the framework by means of case studies and a performance evaluation respectively whilst Section 7 examines reconfiguration aspects. Finally, section 8 analyses related work and section 9 offers our conclusions.

2 MOTIVATION

The taxonomy of ad hoc routing protocols can be broadly broken down into reactive, proactive and hybrid classes. *Reactive* (or on-demand) protocols discover routes to destinations only when there is a need for it. They have low overhead but suffer from route discovery latency. *Proactive* (or table-driven) protocols, on the other hand, continuously evaluate routes from each node to all reachable nodes, and disseminate this information periodically. They exhibit lower latency, but result in higher traffic overhead even if there is no change in network topology. *Hybrid* protocols use various strategies to reduce the overhead associated with flooding the network with control messages: e.g. using hierarchies [6], core-nodes [3], multi-scope operation [4], fish-eye propagation [13, 14] or location-awareness [15].

Several performance evaluation experiments (for example, [16, 17] and [18]) performed within the MANET community have highlighted the differing performance characteristics of various MANET routing protocols. In particular, the average packet delay, packet delivery ratio, routing overhead

and path optimality of these protocols have been observed to vary radically differently with respect to similar sets of conditions (node mobility, network node density and traffic patterns). Reactive protocols generally tend to have low control overhead, can cope with reasonable mobility scenarios but incur some delay during route determination. Proactive protocols, on the other hand, have low average packet delay, high throughput and periodically disseminate routing updates to maintain routes to all nodes so that routes are immediately available when requested. However, unless reduced through the use of limited or efficient flooding techniques, their control overhead poses a major hindrance to scalability in large networks.

Clearly, different protocols are optimised for different conditions/ assumptions, and are thus fit for different application scenarios. In this respect, several protocol optimisation techniques have been proposed to extend the applicability of these protocols. Path accumulation [2], expanding ring search [1], pre-emptive routing [19] and multipath routing [20], efficient/limited flooding [21] are examples of some of the techniques that seek to improve one particular property of the protocol. However, the diversity induced by opportunistic interactions in the MANET domain, makes the deployment of such optimisation solutions ad hoc at best. Also, the resource-constrained mobile devices on which the protocols are deployed, call for lean and efficient communication frameworks where only the cost of the functionality in use needs to be paid for.

Therefore, the opportunity arises for a framework (Manetkit) that simplifies the tasks of expressing MANET protocols and multi-personality protocol configurations, and accommodating additional protocol functionality as plug-ins. In addition, the framework can be used to explore and gain an understanding of the issues involved in dynamic reconfiguration to optimise and hybridise MANET protocols.

3 BACKGROUND

This section presents essential background on the context of our framework. We briefly discuss i) our OpenCOM component model, ii) the notion of component frameworks that we use to provide structure and constraint in component compositions, and iii) our 'open overlays' component framework which supports the development, deployment and dynamic reconfiguration of overlay networks. Our framework builds directly on the latter.

3.1 OpenCOM

OpenCOM [10] is a language-independent lightweight component model designed to support the development of systems software. We use it to provide the basic building blocks into which protocol functionality can be decomposed, and constructed as component configurations. OpenCOM components

are independently deployable units of functionality that may export multiple interfaces and receptacles (receptacles are ‘required interfaces’ that denote a dependency on an interface provided by another component). OpenCOM employs a small runtime kernel that manages the lifecycle of components. It also inherently supports run-time reconfiguration in a generic and principled way through the use of reflection [10].

3.2 Component Frameworks

OpenCOM’s explicit representations of dependencies between components, and its reflection support, are insufficient to structure large scale compositions. On top of these, therefore, we employ the notion of *component frameworks* (hereafter known as CFs) [11] to provide structure to domain-specific component configurations. In essence, CFs are scoped compositions of components that accept plug-in components (including other CFs) that are validated according to constraints specified by the CF to ensure that plug-ins do not produce nonsensical configurations. Since CFs are packaged as components, they can be loaded and unloaded dynamically so that only functionality that is actually instantiated needs to be paid for. As an example of a non-trivial CF, *GridKit* [11] is a CF that provides a complete reconfigurable middleware architecture. It can be used to produce different middleware personalities by instantiating appropriate components and internal CFs at different layers in its architecture. In addition, CFs can be implemented in different programming languages through

To help manage configuration (and dynamic *re*configuration) in CFs we employ a ‘meta component framework’ called Plastik [22]. This uses a general purpose ADL (e.g. ACME [23]) to define legal CF configurations, and employs FOPL-based invariants to ensure that these hold across reconfiguration operations. Crucially, ADL specifications are causally connected at runtime to the OpenCOM kernel so that the invariants can be actively policed. Programmed reconfiguration operations can be specified in Plastik using an *on* (*<dynamic_condition >*) *do* *<actions>* construct that has been added to the ADL.

3.3 The Open Overlay Framework

The open overlays framework [12] is a key sub-CF of the above-mentioned GridKit middleware. Its purpose is to provide a base for building application-level networks.

The framework allows the diversity of overlay protocols to be treated in a consistent manner by adopting an internal structure (referred to as the ‘CFS’ structure; see figure 1) consisting of: i) a *Control* element, ii) a *Forwarding* element and iii) a *State* element. The control element encapsulates the distributed algorithm used to establish and maintain the overlay structure; the forwarding element encapsulates the forwarding strategy employed by

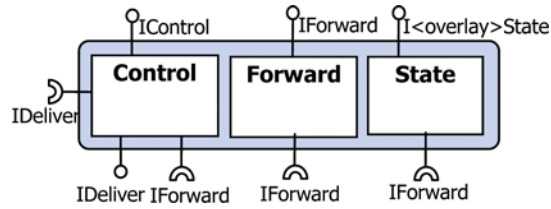


FIGURE 1
The CFS structure of an overlay framework

the protocol; and the state element gives access to generic state such as a nearest neighbour list. Each of the CFS elements implements standard interfaces (shown as small circles in Figure 1, and receptacles (shown as cups) that express a requirement for an interface of the specified type. Building on these, the framework supports the *composition* of CFS structures so that overlays can be stacked on top of each other or composed horizontally. Furthermore, Plastik can be used to ensure that such compositions are suitably constrained. We apply similar architectural principles in our framework discussed next.

4 ARCHITECTURAL OVERVIEW

Manetkit is a component framework (CF) that supports the development, configuration (and potentially dynamic reconfiguration) of ad hoc routing protocols. It supports the development of protocols in multiple programming languages and is generally assumed to run in user space on top of commodity OSs. In terms of protocol design and implementation, it provides the developer with an extensible set of common protocol functionality and tools to configure protocol graphs according to the application needs and environmental contexts.

In more detail, we propose an architectural model that employs run-time deployable software components (based on our OpenCOM system [10]) and CFs to decompose and configure protocol functionality (as shown in Figure 2).

Manetkit comprises of two key sub-CFs as coarse-grain compositional units: i) the System CF which encapsulates common system-related functions; and ii) the ManetProtocol CF which encapsulates protocol-related functions. As illustrated in Fig. 1, an instance of Manetkit running on a mobile node consists of a number of ManetProtocol CFs atop of a System CF (general tree-like compositions are supported; not just stacks). All interaction between the CFs occurs in the form of event-propagation and calls across explicit ‘bindings’ between interfaces (represented as small circles) and so-called ‘receptacles’ (represented as small cups). Receptacles are ‘required interfaces’ that denote a dependency on an interface provided by another component.

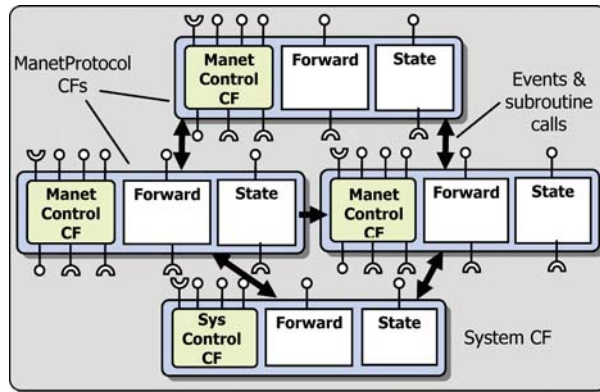


FIGURE 2
The Manetkit framework architecture

4.1 The CFS Design Pattern

To structure the implementation of protocol plug-ins at a finer-grained level, we employ a generic design pattern called the control-forward-state pattern [12] (hereafter referred to as the “CFS” pattern). Using this pattern, each protocol implementation consists of: i) a control element (in Fig. 1, ManetControl in the ManetProtocol plug-in and SysControl in the System plug-in), ii) a forward element and iii) a state element. The pattern corresponds naturally to the domain of MANET protocols: the ManetControl element encapsulates the routing algorithms used to establish and maintain multi-hop routes; the forward element encapsulates the protocol’s send/receive functionality; and the state element gives access to the protocol-specific state such as control message history or a topology table. Each of the CFS elements implements standard interfaces and receptacles.

The CFS pattern encourages both horizontal and vertical composition such that protocol-stacking in Manetkit occurs in a hierarchy of aggregation (again, see Figure 2). For example, an AODV plug-in can be stacked on an MPR plug-in to flood its route requests. It also integrates straightforwardly with higher-level middleware and protocols so that, for example, a service discovery overlay might be stacked on an AODV plug-in to send and route its control messages in the MANET. In the following two sub-sections, we discuss in more detail the System CF and the ManetProtocol CF respectively.

4.2 The System CF

The System CF (shown in Figure 3) performs two main tasks: event handling and the provision of a generic operating system interface.

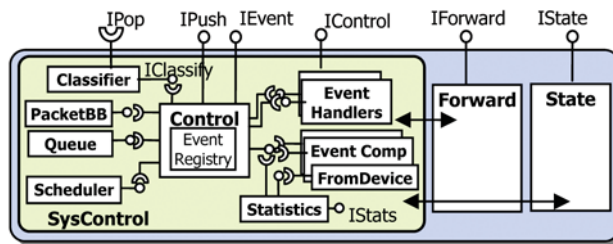


FIGURE 3
The System CF

Event handling: The CF generates two types of events: *control-events* and *information-events*, which are then demultiplexed and forwarded by the Classifier component in its *SysControl* sub-CF. To receive required events, protocol plug-ins in the layer above must explicitly register an interest for such events with the SysControl CF's classifier. This avoids the need for events to cross all the layers in the routing stack. Control-events (generated by the SysControl element) are data structures containing protocol messages parsed from a generalised packet format [24]. To receive this event type, protocols declare their intention to receive network messages via a particular network device and port using Forward component's API functions. These protocol messages are captured using a packet capture library such as libpcap, or packet filters (like Netfilter under Linux or the NDIS intermediate driver under Windows). Information-events, on the other hand, report contextual information (e.g. packet loss, signal strength, and link quality) or reactive routing events to protocol instances in the higher-layers.

Generic system interface provision: To facilitate portability, the System CF acts as a generic surrogate to targeted OS-specific APIs. Its state component provides OS-independent operations to manipulate the kernel route table, initialise the host's routing environment, and query information about the system's network interfaces. Its Forward component provides operations for socket management and defines send/receive primitives for the exchange of protocol messages, that abstract over the use of multiple network technologies. The System CF also providing a platform-agnostic API for thread management.

4.3 The ManetProtocol CF

Each protocol in the MANET routing stack is realised as a CFS-based Manet-Protocol CF that consists of a *ManetControl* sub-CF, a forward component, and a protocol-specific state component. The ManetControl element is the main locus of accommodating MANET protocol diversity, and it also encapsulates a number of areas of commonality, thus enabling coverage of the

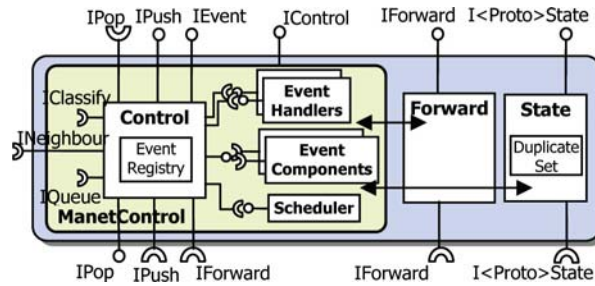


FIGURE 4
A System CF

diverse ad hoc routing protocol taxonomy. The forward and state components are much more specific to individual protocols. Therefore there is less value in providing configurable sub-CFs in those areas.

The ManetControl element (see Figure 4) is responsible for encapsulating a number of common utility tasks and algorithms that many routing protocols share. For example, all need to schedule timers, maintain packet queues and flood control messages (route requests, LSUs) across the network. The simpler tasks are encapsulated as components that are plugged in the CFs. For instance, the Scheduler component is used to register periodic event-generation functions such as Link State Update (LSU) dissemination or timeout functions to expire transient state such as a user-space route table. Because naïve flooding is heavily resource-intensive and thereby limits the scalability of protocols, Manetkit provides a range of pluggable flooding algorithms that includes multi-point relaying (MPR) [26] and the fuzzy-sighted link state algorithms (FSLs) [21]. These are realised as separate ManetProtocol CFs that can be individually plugged below routing protocol CFs in the stack depending on particular sets of circumstances. For example, MPR is used for dense networks whereas FSLs is better in sparse networks of large diameter.

In addition the above, the ManetControl element accepts a number of plug-ins, among which the following three are key:

Control: This is the pluggable control component that implements a particular routing algorithm as a set of event handlers and event producers. It also uses the services of an event queue, an event classifier, a scheduler, and a neighbour discovery protocol CF in addition to the forward and state components. The IPop and IPush interface-receptacle pairs define the upward and downward event routing directions to adjacent ManetProtocol CFs respectively. Since Manetkit supports an open and extensible event-model, the IEvent interface allows developers to extend the set of event types and bind new event handlers to new or existing event types. The Event Registry records semantic information about new event types such as an event type identifier,

the producer component, associated internal handlers and the order they are to be executed and the event's direction of propagation (upwards or downwards). Any unhandled event is passed up or down the protocol graph depending on its direction.

Queue: Queue plug-ins are accepted at the IQueue receptacle and are used by the control component to buffer incoming events for asynchronous event processing if a thread-per-protocol paradigm is adopted by the protocol developer. However, event queuing is optional if the developer wishes to take advantage of the absence of race conditions in a thread-per-event concurrency model. Hence, the concurrency model of the framework is kept orthogonal to its design to provide the choice of either approach.

Neighbour discovery: Neighbour discovery plug-ins (accepted at the INeighbour receptacle) are themselves ManetProtocol CFs that are used to maintain network neighbourhood information. The Neighbour Discovery protocol has the responsibility of reporting link breaks with lost neighbours to the control component for purposes of route invalidation. It also offer a useful means of disseminating information to neighbours; for instance, OLSR [3] piggybacks multipoint relaying (MPR) information on neighbour discovery messages for MPR set advertisements.

5 ILLUSTRATION OF USE

To illustrate the configurability of Manetkit, we now demonstrate its use in building two ad hoc routing protocol configurations, both of which have been implemented in our Java-based Manetkit implementation (see Section 6). In the first instance, we use Manetkit to compose the well-known AODV reactive protocol, showing in the process how it can be extended to accommodate multipath routing (thus illustrating Manetkit's capacity for fine-grained addition of value added behaviour). The second example focuses on a hybridised protocol configuration that combines AODV and OLSR to further demonstrate the flexibility of our framework.

5.1 Implementing and Extending AODV

Figure 5 illustrates our framework implementation of AODV. It consists of an AODV and a Network Neighbourhood ManetProtocol CF layered directly on top of the System CF. To set up the required inter-layer communication all that is required is for: i) the AODV and Network Neighbourhood ManetControl component to be connected to the underlying System CF's event distribution and packet sender services; ii) the higher-layer forward components to be connected to the packer sender; and iii) the AODV state to be connected to

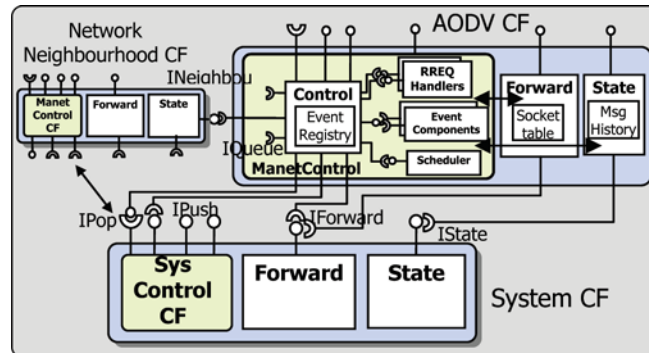


FIGURE 5
AODV protocol configuration

the System CF's state component. Using either packet snooping or packet filtering, the SysControl element detects data packets requiring route discovery and sends RouteDiscoveryEvents to AODV's ManetControl element. Packet queuing occurs in a Queue component at the SysControl element (a particular packet dropping policy can be applied here, or the buffered packets can be re-injected when the System CF is notified by the routing protocol CF of the new route establishment).

In highly dynamic MANETs where link failures and route breaks occur frequently, multi path routing techniques such as AOMDV [20] provide efficient fault tolerance and faster recovery. Using Manetkit, the protocol configuration in Figure 5 can be easily modified for AOMDV operation by i) replacing the RREQ handler to process multiple RREQs, and ii) replacing the AODV State component to accommodate new fields required by the algorithm. Deploying several other optimisation techniques is just as easy.

5.2 A Hybrid AODV/OLSR Deployment

Manetkit can easily be configured as depicted in Figure 6 for hybrid ad hoc routing (by multi-scope operation of AODV and OLSR). Firstly, the OLSR CF is configured to proactively maintain routes to all destinations in the node's zone (defined by a zone radius). It is stacked atop the MPR CF to utilise its efficient-flooding services. Further, whenever a route look up fails in the proactive zone, an inter-zone route discovery event is dispatched by the System CF to the AODV plug-in. Because of explicit event registration, this event does not cross intermediate CFs. The AODV CF then uses the border-casting services of the underlying Border Resolution Protocol (BRP) CF [4] to efficiently forward its RREQs to peripheral nodes. Building this complex, non-vertical routing stack for zonal routing is straightforward; it requires only connecting the protocol CFs and configuring protocol parameters.

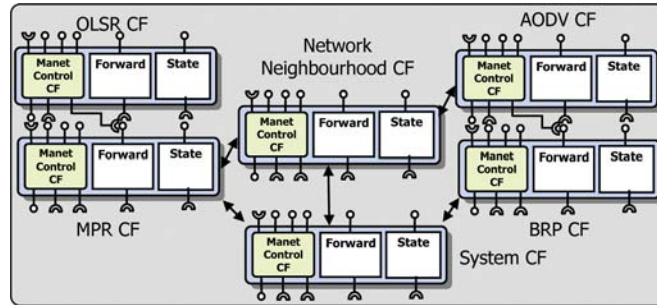


FIGURE 6
Hybrid protocol configuration

6 IMPLEMENTATION AND EARLY EVALUATION

While Section 5 has already evaluated our framework in terms of its configurability and expressibility, we now offer a preliminary evaluation of the overheads of our framework. This is based on a performance comparison between an existing AODV Java implementation (UoB-Jadhoc [26]) and the Manetkit-based AODV implementation that was described above. The overheads we focus on are i) typical performance impact in a multi-node MANET (‘avg route discovery delay’), ii) the overhead of passing packets between layers on a single node (‘time to process RREQ’), iii) per node memory footprint overhead, and iv) the time required to initialise a node.

In Table 1, *avg route discovery delay* measures the overall delay between sending a route request, and receiving and processing a route reply in a small 5-hop MANET testbed. The results demonstrate that Manetkit imposes no major impact on system-wide AODV protocol performance. At a finer-grained level, *time to process RREQ* measures the delay incurred between receiving a route request message from the kernel, and sending it out again. This is a good indicator of the overhead of Manetkit’s componentisation of the packet processing path. Similar numbers are recorded for route reply messages. As the numbers

	UoB-Jadhoc	MANETKit-AODV
Avg route discovery delay	328 ms	345 ms
Time to process RREQ	< 1ms	< 1ms
Memory footprint	107 KB	218 KB (incl OpenCOM)
Initialisation time	68 ms	99 ms

TABLE 1
Evaluation Results

indicate, the differences between the two implementations are minimal to the point of insignificance. In the *memory footprint* metric, Manetkit-AODV incurs a 50% memory overhead over UoB-Jadhoc. This is mainly due to the (necessary) inclusion of the OpenCOM runtime (71 KB). Although this is non-trivial, it is a ‘once-only’ cost in that the same runtime can support multiple protocols and also an overlying OpenCOM-based middleware system and application. More generally, the memory cost is part of a trade-off for the increased flexibility. Finally, the *initialisation time* metric measures the time required to load the protocol modules, configure them and start listening for packets. We find that our CF-based approach does not lead to significant overheads. This metric is also a good indicator of reconfiguration costs in our future work.

7 DYNAMIC RECONFIGURATION

As different protocols are optimised for different conditions and assumptions, the opportunity arises for dynamic reconfiguration to optimise running protocol deployments (e.g. to make them more reactive or proactive) or even to switch the routing strategy altogether. Table 2 indicates some likely possibilities for dynamic reconfiguration that we are currently exploring. The table sets

Condition	Reconfiguration Strategy
Drop in bandwidth	Switch to reactive routing. If large network diameter, use MPR for RREQ flooding.
Frequent transmissions, random source-destination pair and/or bounded delay jitter	If sufficient bandwidth available, switch to proactive routing, else if only subset of nodes involved, use reactive protocol
High packet loss rate and high network churn	Reactive routing: turn on path accumulation or multipath routing Proactive routing: schedule earlier transmission of LSUs
Large network with dense islands	Multi-scoped operation with intra-zone proactive routing and global reactive routing
Hot destinations such as internet gateways or certificate servers	Create a proactive routing zone around destination, or maintain routes to these nodes though route repairs or periodic self-advertisements via RREPs.

TABLE 2
Some Reconfiguration strategies

out reconfiguration strategies together with some corresponding conditions that might trigger such strategies.

In purely ‘local’ terms, reconfiguration amounts to changing protocol parameters and plugging protocol-optimisation sub-frameworks into protocols. For example, in Figure 6 the zone radius can be adjusted as a ‘virtual knob’ à la ZRP to control the mix of proactive and reactive routing, and achieve a desired trade-off between delay jitter, packet loss rate and control overhead. As a coarser-grained example, the Figure 6 scenario can be optimised by plugging in a FSLS flooding CF when the network topology changes to being sparse and large. This increases the proactive zone coverage at a fraction of the maintenance cost involved than if flooding were to be solely performed using MPR.

To realise such strategies, OpenCOM’s reflection facilities [10] already provide good support for ‘local’ reconfiguration (i.e. in individual nodes). In terms of distributed reconfiguration support, the multi-hop nature of MANETs makes it difficult to reliably perform tightly synchronized reconfiguration on all nodes (as is supported, for example, by our existing work on distributed component framework -based reconfiguration support [11]). We are therefore investigating the use of gossip protocols to achieve consensus in the face of message loss and node churn. The gossip protocols can themselves be implemented using the Manetkit framework.

8 RELATED WORK

As mentioned, there have been earlier efforts in the MANET community to develop frameworks with integrated functionality for implementing ad hoc routing protocols. The Ad Hoc Support library [7] and PICA [8] enhance underlying system services and provide MANET-specific APIs such that routing protocols can be developed in user-space. The PICA API is notable for providing multi-platform functionality for process management, memory management for packet queues, socket-event notifications to waiting threads, and network device listing, as well as eliminating platform-related differences in socket APIs, and kernel route table manipulation. However, these systems are restricted to providing programming abstractions for operating system-level services only; they ignore generic routing protocol commonalities that could be reused across implementations.

Click [9] offers a component-based approach to protocol composition in terms of fine-grained elements that are connected together in a packet flow graph. But our framework goes beyond this through its consistent use of the CFS pattern to guide the composition of protocols from fine-grained elements. Also, in a Click router graph fine-grained elements have to be incorporated in the packet forwarding path even if they don’t participate in packet forwarding; e.g. routing tables. In comparison, Manetkit does not require packets to flow

to all plug-in components and sub-frameworks: the packet forwarding and routing functions are kept separate. In addition, Manetkit gains from building on OpenCOM and can leverage its associated services (e.g. its dynamic reconfiguration support [10] and natural integration with higher level middleware services [10, 11]). In terms of dynamic reconfiguration support, Click supports only local reconfiguration in an ad hoc manner in the form of hot-swapping.

Finally, more general protocol frameworks have been proposed in the past (e.g. [29, 30]). These may be suitable for the MANET domain but they are targeted more generally and do not have the MANET-specific features of our design (e.g. the focus on network-level topology management or the network neighbourhood CF). It is also interesting to observe that these frameworks have never been widely deployed. We believe that this is because pre-MANET environments have not been sufficiently rich or diverse to make their use worthwhile. In contrast, due to the inherent diversity of application needs and operating conditions, we expect the MANET domain to continue to generate protocol diversity—and to therefore benefit from the framework approach.

9 CONCLUSION AND FUTURE WORK

Our work has introduced a level of uniformity in ad hoc protocol development by proposing a common ‘CFS pattern’ for expressing diverse routing protocols and integrating them into a wider middleware framework. The CFS pattern exploits the fact that a number of areas of common functionality can be identified across ad hoc routing protocols. Further, the pattern allows protocols to be stacked or composed in a variety of ways to offer a multi-personality routing platform. An added value of our CF approach is that the framework can be reconfigured by loading/replacing protocol mechanisms to suit the current context. This opens the way for run-time protocol hybridisation to improve routing performance.

We have evaluated Manetkit by showing how it can be used to straightforwardly configure two very different protocols (i.e. AODV and a hybridised AODV/OLSR deployment), and by comparing the performance of our AODV implementation with a monolithic alternative.

In the future, a more comprehensive evaluation of our framework will mandate integrating a wider set of protocols. We also plan to build other protocol optimisation mechanisms in a bid to hybridise ad hoc routing protocols and evaluate various reconfiguration strategies. A further step would be to use our framework as a vehicle for investigating a converged ad hoc routing protocol: this is currently a live issue in the IETF’s MANET WG.

REFERENCES

- [1] C. Perkins and E. Royer (2003). Ad hoc On demand Distance Vector (AODV) routing, *Internet Draft*, RFC3561.

- [2] I. Chakeres, C. Perkins (2007). Dynamic MANET On-demand (DYMO) Routing, *draft-ietf-manet-dymo-11*, IETF's MANET WG.
- [3] T. Clausen, C. Dearlove, P. Jacquet (2007). Optimized Link State Routing Protocol version 2, *draft-ietf-manet-olsrv2-03*, internet draft, Feb. 2007.
- [4] Z. J. Haas, M. R. Pearlman and P. Samar (2002). The Zone Routing Protocol (ZRP) for Ad-Hoc Networks, *IETF MANET, Internet Draft*, July '02.
- [5] V. D. Park and M. S. Corson (1997). A highly adaptive distributed routing algorithm for mobile wireless networks, In *Proceedings of IEEE INFOCOM'97*, Kobe, Japan, Apr. 1997, pp. 1405–1413.
- [6] C.-C. Chiang (1997). Routing in clustered multihop, mobile wireless networks with fading channel, *Proceedings of the IEEE Singapore International Conference on Networks (SICON'97)*, pp. 197–211. N. Bhatti, M. Hiltunen, R. Schlichting, and W. Chiu (1998). Coyote: A system for constructing fine-grain configurable communication services. *ACM Transactions on Computer Systems*, 16(4):321–366.
- [7] V. Kawadia, Y. Zhang, B. Gupta (2003). System Services for Ad-Hoc Routing: Architecture, Implementation and Experiences. In *Proceedings of the 1st international Conference on Mobile Systems, Applications and Services* (San Francisco, California, May 05–08, 2003). MobiSys '03. ACM, New York, NY, pp 99–112.
- [8] C. M. T. Calafate, P. Manzoni (2003). A multi-platform programming interface for protocol development, *Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing Proceedings*, pp. 243–249.
- [9] R. Morris, E. Kohler, J. Jannotti and M. F. Kaashoek (1999). The Click modular router. *SIGOPS Oper. Syst. Rev.* 33, 5 (Dec. 1999), pp 217–231.
- [10] G. Coulson, G.S. Blair, P. Grace, A. Joolia, K. Lee, J. Ueyama, T. Sivaharan (2008). A generic component model for building systems software, *ACM Trans. Comput. Syst.* 26, 1 (Feb. 2008), 1–42. DOI=<http://doi.acm.org/10.1145/1328671.1328672>
- [11] P. Grace, G. Coulson, G. Blair, L. Mathy, W.K. Yeung, W. Cai, D. Duce, C. Cooper (2004). GRIDKIT: Pluggable Overlay Networks for Grid Computing, *Proc. Distributed Objects and Applications (DOA'04)*.
- [12] Coulson, G. and Grace, P. and Blair, G. and Porter, B. and Cai, W. and Duce, D. and Cooper, C. and Younas, M. and Sagar (2005). Open Overlay Support for the Divergent Grid, In *Proceedings of the UK E-Science All Hands Meeting 2005*.
- [13] G. Pei, M. Gerla, T.-W. Chen (2000). Fisheye state routing: a routing scheme for ad hoc wireless networks, *Proceedings IEEE International Conference on Communications (ICC) 2000*, pp. 70–74.
- [14] C. Yang and L. Tseng (2007). Fisheye zone routing protocol: A multi-level zone routing protocol for mobile ad hoc networks. *Comput. Commun.* 30, 2 (Jan. 2007), pp 261–268
- [15] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward (1998). A distance routing effect algorithm for mobility (DREAM). In *Proceedings of the 4th Annual ACM/IEEE international Conference on Mobile Computing and Networking* (Dallas, Texas, United States, October 25–30, 1998). W. P. Osborne and D. Moghe, Eds. MobiCom '98. ACM, New York, NY, 76–84.
- [16] S. R. Das, R. Castaneda, and J. Yan (2000) Simulation based performance evaluation of mobile, ad hoc network routing protocols, *ACM/Baltzer Mobile Networks and Applications (MONET) Journal*, pp. 179–189, July 2000.
- [17] J. Broch, D.A. Maltz, D.B. Johnson, Y.-C. Hu, and J. Jetcheva (1998). A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols, In *Proceedings of ACM/IEEE MOBICOM'98*, Dallas, TX, Oct. 1998, pp. 85–97.
- [18] P. Johansson, T. Larsson, N. Hedman, B. Mielczarek, and M. Degermark (1999). Scenario-based performance analysis of routing protocols for mobile ad-hoc networks. In *Proceedings of the 5th Annual ACM/IEEE international Conference on Mobile Computing*

- and Networking* (Seattle, Washington, United States, August 15 - 19, 1999). MobiCom '99. ACM, New York, NY, pp 195–206.
- [19] T. Goff, N. B. Abu-Ghazaleh, D. S. Phatak and R. Kahvecioglu (2001). Preemptive routing in Ad Hoc networks. In *Proceedings of the 7th Annual international Conference on Mobile Computing and Networking* (Rome, Italy). MobiCom '01.
- [20] M. K. Marina and S. R. Das (2001). On-demand Multipath Distance Vector Routing in Ad Hoc Networks. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, pp 14–23.
- [21] J. P. Macker, J. W. Dean (2003). A Study of Link State Flooding Optimizations for Scalable Wireless Networks, *Proc. IEEE Military Communications Conference*, pp. 1262–1267 Vol.2.
- [22] A. Joolia, T. Batista, G. Coulson, and A. T. Gomes (2005). Mapping ADL Specifications to an Efficient and Reconfigurable Runtime Component Platform. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture* (November 06–10, 2005). WICSA. IEEE Computer Society, Washington, DC, pp 131–140.
- [23] D. Garlan, R. Monroe, and D. Wile (2000). ACME: Architectural Description of Component-based Systems, *Foundations of Component-based Systems*, Leavens, G. T., and Sitaraman, M. (eds), Cambridge University Press, pp. 47–68.
- [24] T. Clausen, C. Dearlove, J. Dean, C. Adjih (2007). Generalized MANET Packet/Message Format, *draft-ietf-manet-packetbb-04*, internet draft, Jan. 2007.
- [25] A. Laouiti, A. Qayyum, and L. Viennot (2001). Multipoint relaying: An efficient technique for flooding in mobile wireless networks. *35th Annual Hawaii International Conference on System Sciences* (HICSS'2001).
- [26] K. Kuladinithi, University of Bremen Java-AODV implementation, <http://www.aodv.org>.
- [27] S. Corson, J. Macker (1999). Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations, RFC 2501.
- [28] H. Miranda, A. Pinto, and L. Rodrigues (2001) Appia, a flexible protocol kernel supporting multiple coordinated channels, in *Proc. 21st International conference on Distributed Computing Systems* (ICDCS-21), pp.707–10.
- [29] R. van Renesse, K. Birman, M. Hayden, A. Vaysburd and D. Karr (1998). Building adaptive systems using ensemble. *Softw. Pract. Exper.* 28, 9 (Jul. 1998), pp 963–979.
- [30] P. Grace, G. Coulson, G. S. Blair and B. Porter (2006). A distributed architecture meta-model for self-managed middleware. In *Proceedings of the 5th Workshop on Adaptive and Reflective Middleware (ARM '06)* (Melbourne, Australia, November 27–December 01, 2006). ARM '06, vol. 190. ACM, New York, NY, 3.