

Manetkit: A Framework for MANET Routing Protocols

Rajiv Ramdhany, Geoff Coulson

Computing Department, Lancaster University, Infolab21, LA1 4WA, Lancaster, UK
{r.ramdhany, geoff}@comp.lancs.ac.uk

Abstract

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.

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 [21], DYMO [3], OLSR [6], ZRP [12], TORA [20], CGSR [4] 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 [15] and PICA [2], and more general modular router frameworks like Click [16]—in being much broader in scope (see Section 6).

The design of our framework, which we call Manetkit, builds on our past experience of developing frameworks for middleware systems [9, 10] and overlay protocols [8]. 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 gives an overview of Manetkit’s architecture. Sections 3 and 4 subsequently evaluate the framework by means of case studies and a performance evaluation respectively whilst Section 5 examines reconfiguration aspects. Finally, section 6 analyses related work and section 7 offers our conclusions.

2. Architectural Overview

Manetkit is a component framework 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 [9]) and component frameworks (hereafter, CFs) to decompose and configure protocol functionality (as shown in Fig. 1). CFs are scoped compositions of components that accept plug-in components (including other CFs) that are validated according to constraints specified by the CF designer to ensure that plug-ins do not produce

nonsensical configurations. CFs are themselves packaged as components, so they can be loaded and unloaded dynamically so that only functionality that is actually instantiated needs to be paid for.

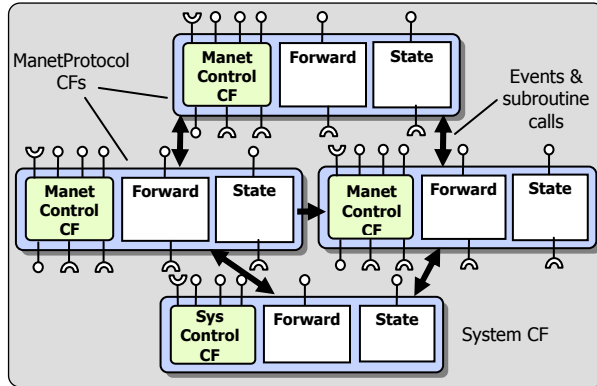


Fig. 1. The Manetkit framework architecture

Manetkit comprises of two key sub-CFs: i) the *System CF* which encapsulates common system-related common 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.

2.1. The CFS Design Pattern

To structure the implementation of protocol plug-ins, we employ a generic design pattern called the *control-forward-state pattern* [8] (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 Fig. 1). 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.

2.2. The System CF

The System CF (shown in Fig. 2) performs two main tasks: event handling and the provision of a generic operating system interface.

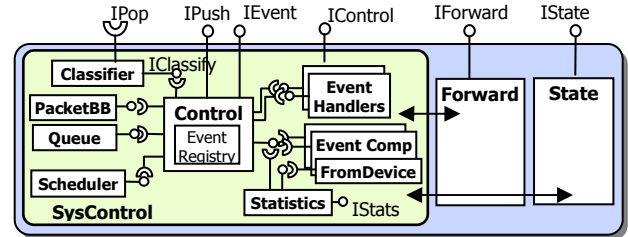


Fig. 2. The System CF

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 [5] using the PacketBB component. 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, are generated by event components to report contextual information (packet inter-arrival time, packet loss, signal strength, and computed statistics) or reactive routing events.

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, its forward component uses a raw IP socket to inject packets back in the kernel output chain just before the packet forwarding function. The System CF also provides a consolidated API for thread and

socket management that transcends platform-specific differences and abstracts over the use of multiple network technologies.

2.3. The ManetProtocol CF

Each protocol in the MANET routing stack is realised as a CFS-based ManetProtocol 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 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 Fig. 3) 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) [12] and the fuzzy-sighted link state algorithms (FSLs) [17]. 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.

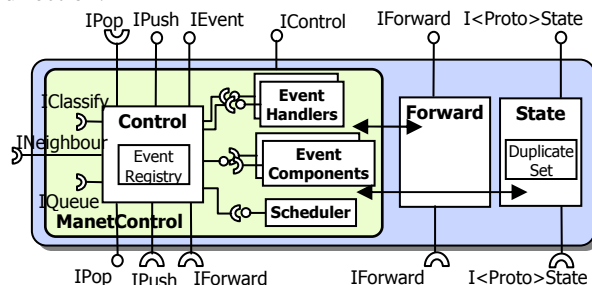


Fig. 3. A ManetProtocol CF

Queue: Queue plug-ins are accepted at the IQueue receptacle and are used by the control component to buffer incoming events 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 2-hop 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 [6] piggybacks multipoint relaying (MPR) information on neighbour discovery messages for MPR set advertisements.

3. 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 4). 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.

3.1. Implementing and Extending AODV

Fig. 4 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 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 [18] provide efficient fault tolerance and faster recovery. Using Manetkit, the protocol configuration in Fig. 4 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.

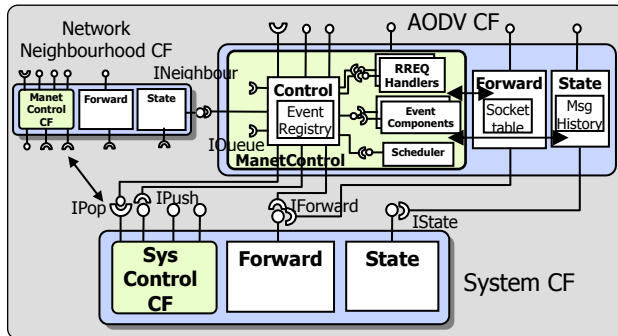


Fig. 4. AODV protocol configuration

3.3. A Hybrid AODV/OLSR Deployment

Manetkit can easily be configured as depicted in Fig. 5 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 bordercasting services of the underlying Border Resolution Protocol (BRP) CF [12] 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.

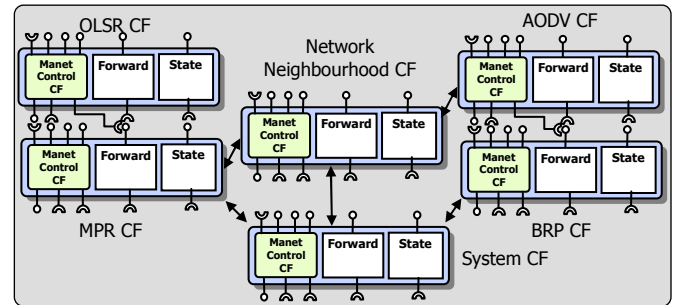


Fig. 5. Hybrid protocol configuration

4. Implementation and Early Evaluation

While Section 3 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 [14]) 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.

Table 1. Evaluation Results

	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

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 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) [9]. 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.

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

Table 2. Some Reconfiguration strategies

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.

In purely ‘local’ terms, reconfiguration amounts to changing protocol parameters and plugging protocol-optimisation sub-frameworks into protocols. For example, in Fig. 5 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 Fig. 5 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 [9] 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.

6. 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 [15] and PICA [2] 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 [16] 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 [13] and natural integration with higher level middleware services [8,10]). In terms 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. [1] [19]). 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.

7. 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] N. T. Bhatti, M. A. Hiltunen et al. Coyote: a system for constructing fine-grain configurable communication services. *ACM Trans. Comput. Syst.*, 16(4):321–366, Nov. 1998.

- [2] C. M. T. Calafate, P. Manzoni, “A multi-platform programming interface for protocol development”, Euromicro Conf. on Parallel Distributed and Network, 2003.
- [3] I. Chakeres, C. Perkins, “Dynamic MANET On-demand (DYMO) Routing”, draft-ietf-manet-dymo-11, IETF’s MANET WG, 18 Nov 2007.
- [4] C.-C. Chiang, “Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel”, IEEE SICON’97
- [5] T. Clausen et al., “Generalized MANET Message Format”, draft-ietf-manet-packetbb-07 internet draft, 2007.
- [6] T. Clausen et al. “Optimized Link State Routing Protocol” version 2, draft-ietf-manet-olsrv2-03.txt.
- [7] S. Corson, J. Macker., “Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations”, RFC 2501, 1999.
- [8] G. Coulson, G. Blair, P. Grace. “Open Overlay Support for the Divergent Grid”, UK All Hands Meeting, 2005.
- [9] G. Coulson, G.S. Blair, P. Grace, A. Joolia, K. Lee, J. Ueyama, T. Sivaharan, “A Generic Component Model for Building Systems Software”, to appear in ACM Transactions on Computer Systems, February 2008
- [10] P. Grace, G. Coulson, et al., “GRIDKIT: Pluggable Overlay Networks for Grid Computing”, Proc. Distributed Objects and Applications (DOA’04), 2004.
- [11] P. Grace, G. Coulson, G. S. Blair and B. Porter., “A distributed architecture meta-model for self-managed middleware.”, International Workshop on Adaptive and Reflective Middleware (ARM ’06), 2006.
- [12] Z. J. Haas, M. R. Pearlman and P. Samar, “The zone routing protocol (ZRP) for ad hoc networks”, Internet Draft, July ’02.
- [13] A. Joolia et al., “Mapping ADL Specifications to an Efficient and Reconfigurable Runtime Component Platform”, IEEE/IFIP Conference of Software Architecture, Nov 2005.
- [14] K. Kuladinithi, University of Bremen Java-AODV implementation, <http://www.aodv.org>.
- [15] V. Kawadia, Y Zhang, B. Gupta, “System Services for Ad-Hoc Routing: Architecture, Implementation & Experiences”, Conference on Mobile systems, applications & services, 2003.
- [16] E. Kohler, R. Morris et al., “The click modular router”, ACM Transactions on Computer Systems, v.18 n.3, Aug. ’00
- [17] J. P. Macker, J. W. Dean “A Study of Link State Flooding Optimizations for Scalable Wireless Networks”, IEEE Military Communications Conference, 2003.
- [18] M. K. Marina and S. R. Das, “On-demand Multipath Distance Vector Routing in Ad Hoc Networks”, IEEE International Conf. on Network Protocols (ICNP), 2001.
- [19] H. Miranda, A. Pinto, and L. Rodrigues, “Appia, a flexible protocol kernel supporting multiple coordinated channels”, In Proceedings of The 21st Int’l Conf. on Distributed Computing Systems (ICDCS-21), 2001.
- [20] V. D. Park and M. S. Corson, “A highly adaptive distributed routing algorithm for mobile wireless networks”, in IEEE Infocom, 1997.
- [21] C. Perkins and E. Royer, “Ad hoc On demand Distance Vector (AODV) routing”, Internet Draft rfc3561, 2003.
- [22] R. van Renesse, K. Birman, M. Hayden, A. Vaysburd and D. Karr, Building adaptive systems using ensemble. *Softw. Pract. Exper.* 28, 9 (Jul. 1998).