A Taxonomy of Protocol Frameworks and Gap Analysis for Adaptive Publish/Subscribe Distributed Realtime Embedded Systems[*]

Wendell Noordhof, Joe Hoffert
Department of Computing Science
The King's University College
9125 50 Street
Edmonton, AB T6B 2H3
wendell.noordhof@lab.kingsu.ca, joe.hoffert@kingsu.ca

ABSTRACT

The growing prevalence of distributed real-time embedded systems in applications such as emergency response, disaster recovery, and ambient assisted living necessitates the use of protocol frameworks to support quality of service requirements and respond to changing environment conditions at runtime. This paper presents a taxonomy that can be used to classify protocol frameworks. The taxonomy includes several features that are relevant for supporting adaptive DRE systems. A brief overview of existing work in the area of protocol frameworks and related network management is provided, and this work is evaluated and classified in terms of the taxonomy. Finally, the paper analyzes the current work on protocol frameworks within the context of adaptive publish/subscribe distributed real-time embedded systems and highlights the gaps found. Our results show that adaptive protocol frameworks are (1) still an area largely addressed by research without standardization and (2) deficient in requirements for adaptive publish/subscribe DRE systems.

Keywords

protocol frameworks, adaptation, distributed realtime embedded, publish/subscribe

1. INTRODUCTION

As the number and complexity of distributed systems continues to grow[6, 9], management of network access and quality of service (QoS) accordingly increase in importance. Some applications need to run in hardware and software environments that change at runtime, and potential problems such as dynamic network topology and congestion need to be addressed. Protocol frameworks can provide the flexibility required to leverage and evaluate standard and custom protocols in diverse configurations and environments. Protocol frameworks can also be used to provide adaptability to a system, via the ability to alter the protocols being used in response to dynamic runtime conditions. This adaptability provides management support for QoS requirements as the environment changes. Moreover, protocol frameworks also increase development and runtime efficiency when utilizing multiple protocols which share common functionality.

Distributed real-time embedded (DRE) systems benefit considerably from the use of protocol frameworks. DRE systems have many current and potential applications, ranging from systems monitoring to communications during search and rescue operations. However, this wide range of applicability comes with an equally wide range of challenges. Many of these challenges arise when the DRE system is being used in a publish/subscribe (pub/sub) scenario, such as distributed monitoring and sensing. For example, increasing the number of publishers and/or subscribers can increase congestion in the network. A protocol framework can help manage the publishers and subscribers entering and exiting the system and thereby support data dissemination requirements.

Pub/sub DRE systems deployed in changing environments while the system is running present additional challenges. For example, for a mobile *ad hoc* network (MANET), the conditions such as signal strength and congestion would be constantly changing, possibly resulting in degraded QoS. To address the dynamicity of the system, the framework must adapt at runtime, reacting to network conditions as they are discovered. Using a protocol framework enables the system developer to respond to these changes in the network environment, ameliorating the challenges they present.

This paper presents a survey of existing work, a classification scheme, and gap analysis for adaptive DRE pub/sub systems. Section 2 discusses the different attributes of a suitable framework for a pub/sub DRE system. Section 3 describes the current state of research in the area of protocol frameworks, as well as relevant network algorithms and technologies, Section 4 classifies these frameworks according to the attributes discussed in Section 2. Section 5 discusses important gaps in the current state of research, and Section 6 provides concluding remarks.

[*]This work is supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Solo Discovery Grant #155709773.

2. TAXONOMY OF PROTOCOL FRAMEWORKS

Adaptive protocol frameworks are useful in a variety of purposes and contexts, such as finding and communicating with nodes in a wireless sensor network and handling cell phone hand-offs from one cell tower to another. This section presents a taxonomy that categorizes the properties common between adaptive protocol frameworks and the properties that differentiate them so that existing frameworks can be compared and contrasted meaningfully.

• **Relevant network layer(s).** Protocol frameworks typically only address certain network layers. For example, some frameworks provide support for developing and managing protocols for the link layer (*e.g.*, link layer for MANETS). Some frameworks provide support for protocols at the network layer (*e.g.*, IP unicast, IP multicast). Other frameworks provide support at the transport layer at the end hosts (*e.g.*, TCP, UDP). Some frameworks provide support at the application layer (*e.g.*, security and management policies for distributed systems).

The emphasis on different network layers ameliorates the complexity of developing protocols at that particular layer. Depending on the network layer of interest some frameworks will not be applicable. However, the functionality provided by frameworks supporting lower level network functionality (*e.g.*, link layer functionality) can be leveraged by frameworks supporting higher level network functionality (*e.g.*, application layer functionality).

• **Accessibility.** Some protocols frameworks only need to run in user space and are generally accessible to any user or developer. These frameworks focus on transport and application protocols since they do not require the permissions needed to access the lower network layers (*i.e.*, network, link, and physical layers). Lack of access to the lower network layers also limits the kinds of protocols that can be developed and the flexibility of designing protocols at any layer of the network stack. The lower network layers already provide certain functionality (*e.g.*, transmitting data from one link to another) that the protocol in user-space can use but not modify.

Some protocol frameworks require that they run in kernel space in order to access and modify IP header information. These kinds of frameworks require that the implementer of the protocols and user of the framework have special access privileges (*e.g.*, root access). Having this kind of privilege allows access to restricted resources which can improve performance and allow greater flexibility in protocol design. However, development in kernel space provides greater opportunities for resources to be misused and for the computer systems to be compromised. Moreover, developers might not always have kernel space access in order to make the necessary protocol framework modifications.

Some protocol frameworks require access to the network infrastructure itself. These frameworks provide functionality below the network layer (*e.g.*, at the link layer) and so need to be able to run in and access the network elements (*i.e.*, routers, switches). Like frameworks that run in kernel space and require special permissions for deployment, these frameworks require access to the network elements. These elements also need to support modification and extension of the software running in them.

• **Static vs. dynamic adaptability.** Some protocol frameworks have been developed to support modification of protocols during development time. For these frameworks the application developer makes code changes to modify the protocol that is being used. This approach provides for more predictability in the deployed framework and applications using the framework. For example, the amount of resources needed (*e.g.*, memory, network bandwidth) can be better predicted and managed since the protocol used is known at deployment time.

Some protocol frameworks allow modification of protocols while the system is running. These frameworks support multiple protocols at deployment time and are able to handle switching from one protocol to another or making modifications to the existing protocol to change its behavior. These types of frameworks allow greater protocol flexibility for an executing system. These frameworks also support adaptive behaviors as the operating environment changes.

Protocol frameworks that support dynamic modifications can also be split into frameworks that provide fine-grained transitions and frameworks that provide coarse-grained transitions. Frameworks that provide coarse-grained transitions typically suspend all functionality while the transition is made from one protocol to another. Frameworks that provide fine-grained transitions will support incremental modification of the protocols (*e.g.*, supporting functionality common between the protocol being transitioned to and the protocol being transitioned from). These frameworks can also allow both the protocol being transitioned from and the protocol being transitioned to to run at the same time during the transitional period.

• **Supported implementations.** Some adaptive protocol frameworks have been developed purely as research projects and proofs of concepts. These frameworks provide insights into the possibilities of adaptive protocol frameworks. These frameworks also allow researchers the freedom to explore areas that have not been the area of focus for industrial or commercial uses.

Some adaptive protocol frameworks have been developed with industrial and commercial challenges in mind and have supported implementations. Therefore, these frameworks are more likely to be used in commercial and industrial applications. Correspondingly, the commercial and industrial sponsors are motivated to keep these frameworks up to date with the latest hardware and software changes (*e.g.*, upgrades of CPU, operating system, and network elements).

• **Events vs. interfaces for protocol composition.** Some protocol frameworks are architected for protocol composition and development via inheritance of programmatic interfaces. These frameworks provide a set of classes that protocol developers can use to implement and/or extend the framework's base functionality. This interface approach makes it easier to manage the protocols that have been developed since the behavior of these protocols can be analyzed statically by evaluating the code. The approach also lends itself more naturally to a request/response or client/server protocol since the method invocations of the interfaces inherently leverage the client/server paradigm.

Some protocol framework architectures provide publication and subscription of events to connect the flow of data for a protocol. These frameworks predefine the events that the protocol developer can leverage to connect functionality together to produce the desired protocol. Moreover, these frameworks make dynamic composition of protocols easier and provide greater flexibility in the composition of protocols.

However, frameworks that leverage publication and subscription of events make analysis of the protocol composition harder. The developer must manage the flow of events (*i.e.*, connect the publication and subscription of the appropriate events) to ensure correct protocol behavior. Keeping track of the flow between protocol modules that comprise a protocol can be challenging since any event can be published or subscribed to by any module. In addition, if the framework supports dynamic reconfiguration of event subscriptions, the flow of events becomes harder to track and manage.

• **Dependence on other technologies.** Some protocol frameworks incorporate third party technologies to provide functionality. These frameworks are able to leverage previous work which provides for faster development or higher level abstractions. These frameworks also introduce dependencies on the supporting technologies. Whenever updates are required for the framework, updates for the supporting technologies are also needed.

Some frameworks are self-contained. These frameworks provide all the functionality themselves and do not require third party technology. These frameworks eliminate the dependencies on third party technology which can increase the complexity of management and development. These frameworks typically only need support from language interpreters or compilers.

• **Support for custom protocols.** Some protocol frameworks provide building blocks or protocol modules so that custom protocols can be constructed and supported within the framework. At a minimum, these frameworks provide a base level of functionality such as sending out and receiving data from the network. These frameworks also provide support for developing custom protocol modules as well as support for connecting the modules together to determine the protocol functionality. The modules can be connected together either via events or common interfaces.

Some frameworks are designed to resolve challenges with currently existing protocols rather than support the creation of new custom protocols. These frameworks provide a common interface for different protocols or protocol modules to raise the level of abstraction depending on the development environment. These kinds of frameworks are also developed to decrease overhead and optimize performance. This abstraction then makes application development easier since the details of a particular operating platform are hidden.

3. PROTOCOL FRAMEWORKS

This section provides a brief overview of various protocol framework approaches considered. This section highlights distinguishing aspects of each approach. This overview is leveraged in Section 4 where the paper taxonomizes the approaches based on the categories described in Section 2.

• **HORUS.** HORUS [14] is a network group communication framework, designed for system, protocol, and configuration flexibility. HORUS can be configured to provide various services such as encryption or reliable transmission. These configurations allow application developers to easily extend the abilities of their programs simply by routing network access through HORUS. This ease of configuration prevents time and energy being wasted by unnecessarily rewriting existing code.

• **MANET Service Discovery.** Existing service discovery solutions are primarily designed for wired networks and consequently are unsuitable for MANETS, due to resource constraints and the lack of central infrastructure. The solution discussed by Florés-Cortes *et al.* [5] involves discovering common elements of existing service discovery middleware and consolidating these elements into a component framework. The component framework reduces the cost of each protocol via component and code reuse, and greatly eases the burden of configuration.

• **Network Processors.** Yong Jun *et al.* [16] work towards developing a software framework that would allow for network processors (NPs) to be easily programmed. NPs have an extremely high level of architectural heterogeneity, incorporating general purpose and specialized processors, varying memory interactions, and severe resource and performance constraints. The focus is on maintaining conceptual uniformity across platforms while also taking advantage of the unique characteristics of different hardware configurations to maintain optimal performance for the protocols used.

• **Service Interface.** Rütti *et al.* [13] focus on optimizing the protocol frameworks themselves. They put forward the idea that service interfaces would allow protocol frameworks to be more effective than the more commonly used event-based abstraction. The main benefit of service interfaces would be reducing the burden on programmers, as the services restrict the way different protocol modules can be plugged in. This restriction prevents common network programming errors, such as subscribing a module to an incorrect event. In addition, dynamically swapping protocols is made easier because a module can be monitoring the service through which communications pass in order to determine the optimal time to switch.

• **Altitude-Based Architecture.** Another approach is given by Chen *et al* [3]. They address the limitations of the current network stack in terms of inter-layer communication and the way this reduces extensibility. Their proposed replacement, Altitude Based Architecture (ABA) consists of a *vertical* element, the Protocols Administrator (PA), to which all the included protocols are attached, according to their layer's altitude and their protocol ID. The PA routes frames between protocols, allowing for easy insertion of new elements to extend the framework's functionality.

• **Distributed Protocol Stacks.** Kliazovich and Granelli propose distributed protocol stacks (DPSs) [7] which are a response to the increasing heterogeneity of modern networks. As MANETs and distributed systems become more important and widely deployed, the traditional TCP/IP network stack becomes less effective. DPSs use cross-layering to enable a greater degree of intra-stack communication, which increases the potential flexibility of each individual layer. In addition, DPSs are designed to function with the assistance of agent-based networking, wherein discreet functional blocks are removed from the network stack and deployed in routers or switches, allowing easier performance of tasks such as network congestion detection or packet acknowledgment.

Kliazovich and Granelli argue that the TCP/IP network stack is becoming less well suited to the architecture of the Internet as heterogeneity becomes the new norm. One of the solutions they discuss is cross-layering, wherein network layers are less atomic and more involved with each other's functionality. Another is agent-based networking, in which software agents external to the stack can be used for such tasks as congestion detection or caching. Both approaches involve abandoning or modifying the traditionally rigid network stack.

• **EPFramework.** Xu *et al.* [15] focus their efforts on developing a network stack architecture called EPFramework to optimize the operation of embedded systems. They propose a system abstraction layer, which would standardize system calls across different systems, making the framework more portable. Another feature is the connection layer, which serves to improve maintainability, manage multiple connections to the stack and enable access control. This work also includes a communication interface layer, which smooths transactions between the stack and the application layer. This feature helps manage the heterogeneous nature of embedded systems. An implementation of SocketLib is included to optimize interactions with the BSD socket API.

• **MANETkit.** MANETkit [12] uses component frameworks (CFs) to define different ad hoc routing protocols. The CFs are built using the Control Forward State design pattern, meaning that each CF contains a *control* element that manages forwarding rules, a *forward* element that manages interactions with the network and other CFs, and a *state* element that maintains relevant protocol-specific state information. A system CF manages interaction with the operating system and handles events, and MANET protocol CFs serve to define the protocols. Plugins can be used to augment the functionality of protocols as needed, and these are also implemented with ManetProtocol CFs.

• **λMAC framework.** The λMAC framework [10] is intended to maintain a high level of uniformity between different implementations of medium access control (MAC) protocols. MAC protocols are responsible for packet transmission and acknowledgment, performed in similar ways across all different MAC protocols. Consequently, having each protocol implement this functionality results in significant code duplication and potential for bugs. The λMAC framework handles these functions, leaving individual MAC protocols primarily responsible for transmission timing, which is the most heterogeneous feature of MAC protocols. The increased uniformity granted by the λMAC framework allows for significantly easier protocol swapping and adaptation.

• **Dynamic Service Replication.** Service replication is a method used to make network services more scalable and likely to be available at any given time. Unfortunately, replication also significantly complicates reconfiguration, either introducing service inconsistency or resulting in a period of unavailability. Dynamic Service Replication (DSR) [2] attempts to eliminate these problems while also avoiding the failing of other solutions in adding steady-state complexity to reduce reconfiguration complexity. The reconfiguration process of DSR roughly follows the following sequence: suspend initial configuration, take a snapshot of the current configuration's state, transfer the state to the new configuration, resume operating with the new configuration.

4. CLASSIFICATION OF FRAMEWORKS

This section of the paper uses the taxonomy developed in Section 2 to classify the frameworks presented in Section 3. This classification helps to clarify the properties of existing protocol frameworks.

• **HORUS.** HORUS is mainly an application layer framework, though it does support some functionality from lower layers. It can be embedded in the kernel of a system, run in user space, or split between user and kernel space. HORUS also has the ability to be reconfigured dynamically. However, this reconfiguration is done at a very coarse-grained level. HORUS suspends all protocol services while the reconfiguration is performed. Moreover, HORUS has a dependency on the Ocaml [8] programming language.

• **MANET Service Discovery.** The MANET service discovery framework designed by Florés-Cortes *et al.* is also based in the application layer, and it operates primarily in user space. This framework does not have dynamic reconfiguration, needing a restart to implement new settings. The discovery framework requires an underlying component technology such as OpenCOM [4] to implement its protocols.

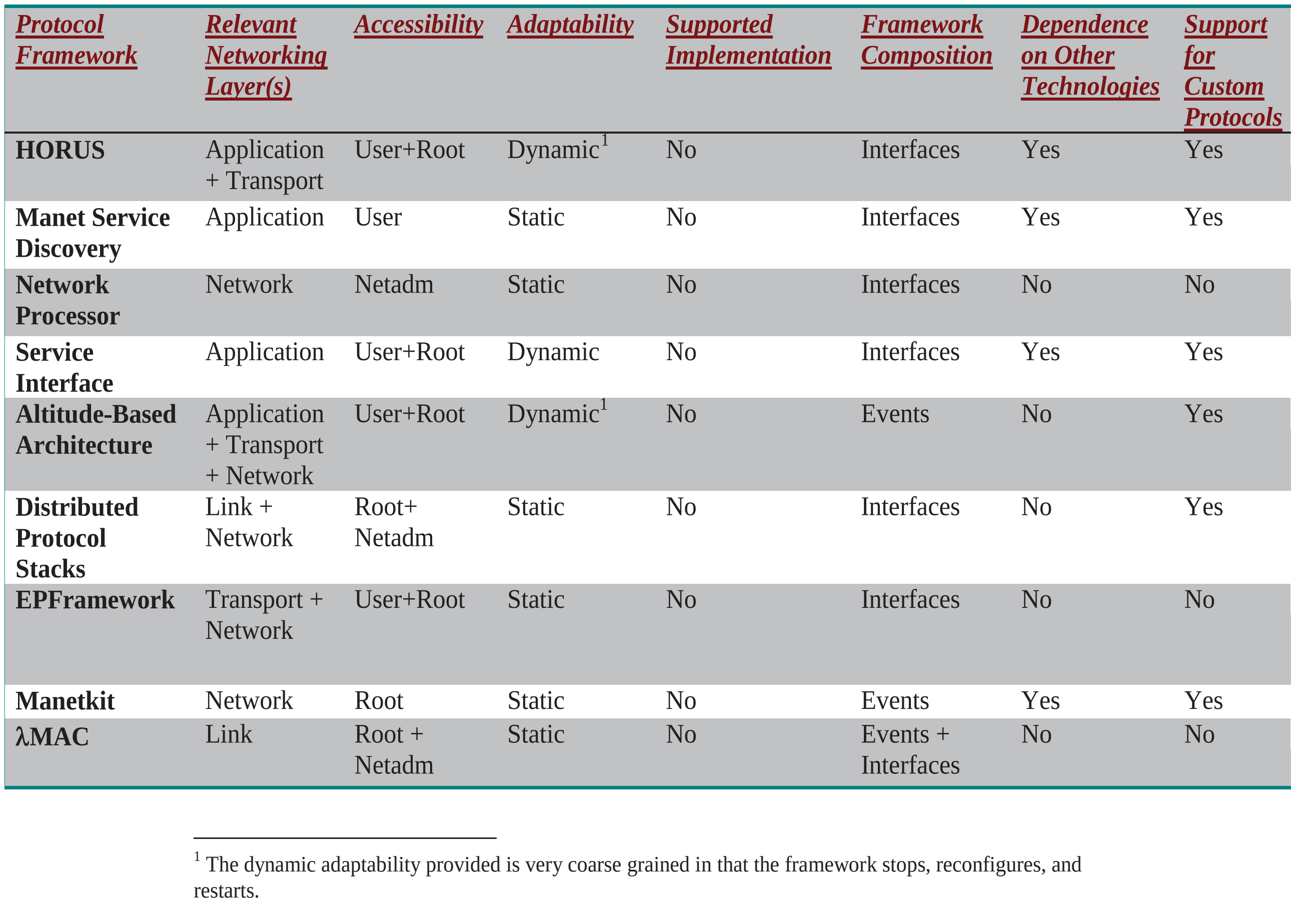
• **Network Processors.** This is a programming framework designed primarily around the link and network layers. It has only static adaptability and is accessible by a network administrator. Due to its low level nature, the network processor framework is not dependent on any other specific technologies. Since the framework is primarily intended as a network processor abstraction, it does not have explicit support for customized protocols.

• **Service Interface.** The service interface framework is primarily used in the application layer, though both users and the kernel can take advantage of it. The service interface framework has support for customized network protocols since protocol modules can interact with any other protocol module to create the protocol with the desired properties. Service interfaces allow for an increased level of dynamic adaptability in protocol frameworks since protocol modules can be dynamically bound together to produce a protocol. SAMOA, which is an experimental framework based on the service interface approach, requires the Java Party [11] and JDOM [1] technologies.

• **Altitude Based Architecture.** Altitude-based architecture (ABA) focuses on communication between stack layers so it cannot be classified in a single layer. However, application, transport, and network layers are the main focus of this research. ABA has a very course reconfiguration granularity, requiring session managers to set up each session individually. Since there is no publicly available implementation, it is difficult to evaluate ABA's dependencies. ABA supports construction of customized protocols.

• **Distributed Protocol Stacks.** The distributed protocol stacks architecture operates on the link and network layers, allowing for greater consistency across network components like routers and switches. It can be manipulated by a network administrator or be integrated into an operating system kernel. However, this architecture has only static reconfigurability since its emphasis is less on protocols and more on distributed processing. The interface-based design allows for customized protocols. No publicly available implementation is provided with this research.

• **EPFramework.** EPFramework is primarily a transport layer framework, with some activity taking place on the network layer. Since its goal is to abstract away the kernel functions, elements of EPFramework are accessible to both the user and the kernel. However, this goal was pursued to the exclusion of adaptability, so EPFramework can only be reconfigured statically. EPFramework uses interfaces to provide access to the functions it is abstracting. It does not explicitly rely on any third party software and has no built in support for custom protocols.



Protocol Framework	Relevant Networking Layer(s)	Accessibility	Adaptability	Supported Implementation	Framework Composition	Dependence on Other Technologies	Support for Custom Protocols
HORUS	Application + Transport	User+Root	Dynamic[1]	No	Interfaces	Yes	Yes
Manet Service Discovery	Application	User	Static	No	Interfaces	Yes	Yes
Network Processor	Network	Netadm	Static	No	Interfaces	No	No
Service Interface	Application	User+Root	Dynamic	No	Interfaces	Yes	Yes
Altitude-Based Architecture	Application + Transport + Network	User+Root	Dynamic[1]	No	Events	No	Yes
Distributed Protocol Stacks	Link + Network	Root+ Netadm	Static	No	Interfaces	No	Yes
EPFramework	Transport + Network	User+Root	Static	No	Interfaces	No	No
Manetkit	Network	Root	Static	No	Events	Yes	Yes
λMAC	Link	Root + Netadm	Static	No	Events + Interfaces	No	No

[1] The dynamic adaptability provided is very coarse grained in that the framework stops, reconfigures, and restarts.

Figure 1: Classification of Protocol Frameworks

• **MANETkit.** MANETkit's focus on routing protocols means that MANETkit is largely restricted to the network layer, as well as being entirely within the domain of the kernel. MANETkit has only static reconfigurability and is configured via publication and subscription of events. MANETkit relies on OpenCOM [4] for its component frameworks and provides support for custom protocols. The protocols can be configured at run-time, and are built with the same modular components as MANETkit itself. MANETkit is currently not publicly available.

• **λMAC framework.** λMAC operates in the link layer, as it is an enhanced version of existing MAC addressing protocols. Its functions are root access only and it has static reconfigurability. λMAC uses both events and interfaces and does not provide support for custom protocols. It is intended to unify existing MAC protocols rather than support new or custom ones. λMAC is currently not publicly available, and its software dependencies are not explicitly defined.

5. ANALYZING GAPS

Certain functionality is desired for protocol frameworks that support adaptive pub/sub DRE systems. Below are listed important gaps in the current research in this area.

• **Timeliness.** Timeliness is a concern for DRE systems. In particular, the amount of time taken to complete any functionality must be bounded. Adaptive DRE pub/sub systems are therefore concerned with (1) the responsiveness of the program or framework to adapt to a given environment, and (2) the timeliness of the data that the network protocols provide. However, none of the frameworks evaluated are designed to explicitly address the bounded timeliness needed for DRE systems and do not provide latency guarantees.

• **Transition Management for QoS Optimization.** Most of the frameworks discussed in Section 3 placed very little emphasis on protocol and configuration transitions during system execution. The majority of the frameworks reviewed have only static reconfiguration. The frameworks that provided dynamic reconfiguration did not address how to best transition from one protocol to another.

With adaptive DRE systems, QoS concerns are as important as functional concerns. For a given environment one protocol can provide substantially better QoS than another protocol. Moreover, QoS can be impacted by how a protocol framework transitions from one protocol to another. For example, if the system needs to transition between protocols, the common functionality between the two protocols can be maintained to reduce the transition time. Additionally, the parts of the protocols not in common can be turned off and on at different times to maximize QoS (*e.g.*, starting up the new protocol while the old protocol is still running). None of the frameworks surveyed provide this needed level of transition management.

• **Interface Standardization.** The reviewed protocol frameworks have taken several different approaches to interfacing with the system developer. While the reviewed research is advancing the development of protocol frameworks, the lack of common interfaces impairs the development of systems that use the protocol frameworks. Standardized interfaces would ease the development of systems that utilize protocol frameworks and eliminate the timeconsuming and error-prone learning curve for using various protocol frameworks. Standardization of programmatic interfaces between layers would also be beneficial.

• **Group Communication Support.** Group communication support is an essential feature when dealing with DRE systems, as their primary benefit comes from their ability to disseminate information from a variety of sources. This benefit is nullified if the system is splintered into individual elements. Consequently, a protocol framework intended for a pub/sub DRE context must include some sort of group communication support.

6. CONCLUDING REMARKS

Our findings reflect two primary problems with existing work. The first problem is a lack of existing support. Several projects fill some of the gaps we have identified, but no project fills all the gaps. Moreover, most projects are not actively maintained. In order for progress to be made in the area of adaptive networking, it is important that the development of software keep pace with the growth of the industry at large.

Secondly, the existence of a standard would be greatly beneficial to expanding work on adaptive networking software, especially within specific network layers. Currently, projects pursue their goals by whatever method those involved perceive to be the best. This allows for relatively quick progress to be made, but inhibits integration of elements from multiple different projects. This shortcoming will become more pronounced as relevant software and concepts appear. Hence, the lack of standards in this area is a major hurdle standing in the way of commercialized adaptive networking.

Overall, we found several pieces of work related to protocol frameworks. These ranged from specific protocol implementations to overhauls of the entire network stack. Our taxonomy identifies some strengths of existing work, including support for non-standard protocols. However, we also reveal several important but absent features and areas for future work, most notably a lack of fine grained control over transitions from one protocol to another.

7. REFERENCES

[1] Wes Biggs and Harry Evans. Simplify xml programming with jdom, May 2001.

[2] Ken Birman, Dahliai Malkh, and Robbert van Renesse. Virtually synchronous methodology for dynamic service replication. Technical report, Microsoft Research Silicon Valley, 2010.

[3] Tian Chen, Liu Wenyu, Wang Yi, and Huaien Luo. A uniform host protocol framework planning to change. In *VTC Spring'08*, pages 2730–2734, 2008.

[4] Geoff Coulson, Gordon Blair, Paul Grace, Francois Taiani, Ackbar Joolia, Kevin Lee, Jo Ueyama, and Thirunavukkarasu Sivaharan. A generic component model for building systems software. *ACM Trans. Comput. Syst.*, 26(1):1:1–1:42, March 2008.

[5] Carlos A. Flores-Cortés, Gordon S. Blair, and Paul Grace. A multi-protocol framework for ad-hoc service discovery. In *Proceedings of the 4th international workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC 2006)*, MPAC '06, pages 10–, New York, NY, USA, 2006. ACM.

[6] Yi Huang and D. Gannon. A comparative study of web services-based event notification specifications. In *Parallel Processing Workshops, 2006. ICPP 2006 Workshops. 2006 International Conference on*, pages pp. 7–14, Los Alamitos, CA, USA, August 2006. IEEE Computer Society.

[7] D. Kliazovich and F. Granelli. Distributed protocol stacks: A framework for balancing interoperability with optimization. In *Communications Workshops, 2008. ICC Workshops '08. IEEE International Conference on*, pages 241 – 245, 2008.

[8] Yaron Minsky. Ocaml for the masses. *Commun. ACM*, 54(11):53–58, November 2011.

[9] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer New York, New York, NY, USA, 3rd edition, 2011.

[10] Tom Parker, Gertjan Halkes, Maarten Bezemer, and Koen Langendoen. The λmac framework: redefining mac protocols for wireless sensor networks. *Wirel. Netw.*, 16(7):2013–2029, October 2010.

[11] Michael Philippsen and Matthias Zenger. Javaparty - transparent remote objects in java. *Concurrency: Practice and Experience*, 9(11):1225–1242, November 1997.

[12] Rajiv Ramdhany, Paul Grace, Geoff Coulson, and David Hutchison. Manetkit: supporting the dynamic deployment and reconfiguration of ad-hoc routing protocols. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '09, pages 1:1–1:20, New York, NY, USA, 2009. Springer-Verlag New York, Inc.

[13] Olivier Rütti, Paweł T. Wojciechowski, and André Schiper. Service interface: a new abstraction for implementing and composing protocols. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 691–696, New York, NY, USA, 2006. ACM.

[14] Robbert van Renesse, Kenneth P. Birman, and Silvano Maffeis. Horus: a flexible group communication system. *Commun. ACM*, 39(4):76–83, April 1996.

[15] Hongzhe Xu, Xiaohui Peng, Li Yue, and Chen Ming. Research in a framework of embedded network protocol stack and application. In *Proceedings of the 2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application - Volume 01*, PACIIA '08, pages 768–771, Washington, DC, USA, 2008. IEEE Computer Society.

[16] Wang YongJun and Huang QingYuan. Research of flexible protocol development software framework based on network processor. In *Proceedings of the 2006 International Conference on Hybrid Information Technology - Volume 01*, ICHIT '06, pages 270–277, Washington, DC, USA, 2006. IEEE Computer Society.