A Router Architecture to Achieve Link Rate Throughput in Suburban Ad-Hoc Networks

Muhammad Mahmudul Islam, Ronald Pose, and Carlo Kopp

School of Computer Science and Software Engineering, Monash University, Australia
{sislam,rdp,carlo}@mail.csse.monash.edu.au

Abstract. Static nodes, e.g. houses, educational institutions etc, can comprise ad-hoc networks using off-the-self wireless technologies with a view to bypass expensive telecommunication solutions. A suburban ad-hoc network (SAHN) aims to provide such an inexpensive broadband networking alternative for communities of cooperating users using wireless medium. There exists a plethora of efficient routing solutions for ad-hoc networks where nodes are mobile. However, less attention has been paid towards optimizing these protocols and developing a real routing system for ad-hoc networks where nodes are not mobile. In this paper we have made analyses of various router architectures and outlined a design framework to perform routing tasks in the SAHN efficiently. We have also presented a survey result for choosing a feasible realtime operating system for our development and deployment purposes.

1 Introduction

The ever increasing trend towards huge amounts of data transfered at high speeds through the Internet has inspired researchers to come up with many new efficient networking technologies. Unfortunately most of the new technologies offering high rates of data transfer require costly infrastructure and high service charges which are only feasible for large educational institutions, governmental organizations, companies and research groups. People in small offices, companies and homes can only enjoy similar performance at great cost. Usually these facilities are only available in close proximity to service providers. Many voluntary networking groups[1] have been formed to provide wireless internetworking facilities by connecting households, schools, community centres and local businesses together at low initial costs and almost no service charges. But these solutions are threatened by unauthorised intrusions. Moreover participants in a community have to rely on centralised routing nodes for intercommunication. This results in performance bottlenecks as well as inefficient use of aggregate network capacity. As a consequence these solutions are still less attractive than the traditional and costly solutions provided by various telecommunication service providers. It can even be argued that Nokia's wireless broadband solution (Nokia RoofTop) for residential users, which has an optimized IP protocol stack with custom-built OS for routing, may result in marginal performance in ad-hoc wireless networks. To alleviate these expensive, oversubscribed, area limited

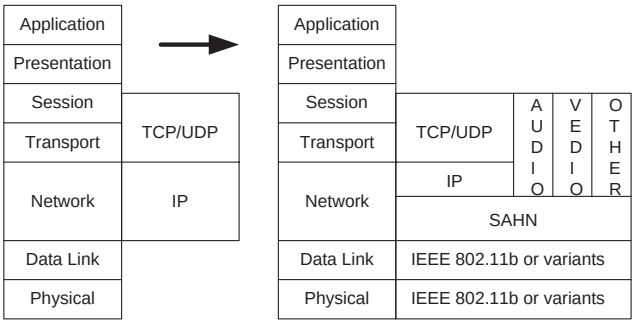

Fig. 1. The SAHN protocol stack in the OSI model [3]

and low secured solutions, a networking framework has been proposed termed the 'Suburban Ad-Hoc Network' [2] or SAHN. The SAHN is a low-maintenance, decentralized, cooperative wireless networking architecture offering low cost internetworking solutions among its users. The inherent symmetric throughput in both upstream and downstream channels at reasonably high rates allows the facility to provide traditional costly broadband throughput at low cost. The security scheme at the network layer is particularly appealing to security conscious business users. Additionally the wireless interconnecting property makes the SAHN suited to extend the Internet infrastructure to areas of inadequate wired facilities.

Each SAHN node is capable of authenticating neighboring nodes to participate in the network. Each node also acts as a dynamic router to discover and maintain routes itself. Initial investigation showed that, the SAHN routing protocol shares the properties of both ad-hoc on-demand and static table driven routing protocols. Notably, the protocol adopts the idea of keeping the neighbor information up-to-date like any of the static table driven routing protocols. On the other hand to find a route to an unknown node, as well as to maintain it, it adopts an on-demand route discovery and maintenance mechanism derived from the Dynamic Source Routing (DSR)[4] protocol. For data transmission over known routes with sufficient QoS attributes, the SAHN routing protocol exploits mixed principles of DSR and the Ad-hoc On Demand Distance Vector Routing (AODV)[5] protocol. As the SAHN does not carry source routes in each data packet, a large network overhead can be eliminated in networks with many nodes[5]. The motivation to adopt a hybrid routing protocol with certain quality of service metrics and resource access control capabilities, is to eliminate the shortcomings of any individual protocol [3][6][7].

The SAHN project aims to come up with a working solution for real ad-hoc networks. As simulation alone of the proposed routing protocol is not enough, we should design an efficient hardware architecture for it to achieve link rate throughput at each node. We should also select a suitable development and

target platform, so that the software development cycle is minimized with least efforts and costs. We have addressed these issues in this paper.

We have organized our paper as follows. In Section 2, we have made some analyses of available design approaches to adopt for the SAHN router. We have also outlined the SAHN routing engine in this section. Finally in Section 3, we have done a requirements analysis as well as a survey to facilitate the choice of a suitable operating system to support our routing protocol.

2 SAHN Router Architecture

Packet processing and switching in a router is always a critical job in terms of time and memory. Without proper hardware and software design a router can be a bottleneck. Proper design means to isolate the time critical packet processing with the non-time critical ones. At the same time it is essential to employ appropriate hardware for the tasks so that they can be processed faster and if possible independent tasks in parallel. Here we will present the hardware architecture and the associated implementation framework to achieve link rate throughput in each of the SAHN nodes.

To begin our analysis, we should know what are the basic components and their related tasks of any router. These are

- Several network interface cards attached to network media to receive and transmit packets
- Some processing modules or forwarding engines to validate incoming packets, to create and maintain routing tables, to update packet header information and finally to encrypt and transmit them through appropriate outbound interface cards
- Buffering modules to hold packets, routing tables etc.
- An internal interconnecting unit like a bus or a switch fabric to enable communication among various working modules.

We can summarize that, a router consists of some tasks to perform routing, some working modules to perform these tasks and interconnecting fabrics to enable intercommunications among various modules. In the following subsections, we will discuss these in detail with respect to the SAHN.

2.1 Routing Tasks

The SAHN router follows the basic tasks of any generic router. Moreover, it has to accomplish some other tasks specific to the SAHN environment. Generally the SAHN router has to carry out the following responsibilities:

1. Receiving a packet at the interface card, the header is separated, classified, decrypted and validated. Taking only the header for the rest of the processing enables us to work faster with a small amount of memory.
2. Then a path entry is searched against the routing information available in the cache or the routing table.

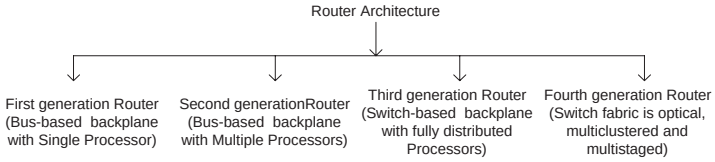

Fig. 2. Classification of router architecture [8][9][10]

3. If the destination path is not found, a route discovery takes place according to the SAHN routing algorithm.
4. If the destination is reachable, the next hop towards the destination and the interface through which the next node is reachable is determined and the packet header information is updated and the level2 header is encrypted for the next hop. Then the level2 and level1 headers are prepended with the rest of the packet.
5. At last the packet is transmitted through the interface card connected to the next node.

In order to achieve router throughput close to the data rate of the interface cards, we have to isolate the time critical tasks (directly related to packet forwarding) from the non-time critical ones and employ appropriate hardware components with an efficient interconnection or switching fabric. A close investigation into the above steps reveals that except from step 3, all other steps can be considered as directly related to forwarding a packet in the SAHN routing module. Step 3 along with some other tasks like route maintenance, providing QoS, route management and error control are not done on a per packet basis. For example route discovery is only needed if the route is not present in the route table/cache. In such case the corresponding packet can be buffered until the route is found. In the mean time all other packets received with known routes can be processed. Best router throughput can be achieved if packets are handled by multiple heterogeneous processing modules (both hardware and software) where each of them specializes in a specific task and work simultaneously where possible.

2.2 Interconnection Fabric

A well designed switch fabric is essential for non-blocking interconnection of the critical components with much higher capacity and speed. The most common switch fabrics used in routers have been discussed elaborately in [8], [11] and [9]. Here we will present an analyses of these switch fabrics before selecting one for our purpose.

A backplane interconnection fabric connects ingress ports with egress ports. Ingress and egress ports are combinations of incoming/outgoing line cards, forwarding engines etc. A well designed intercommunicating backplane switch fabric is very important as it has the effect on overall system throughput. Even though

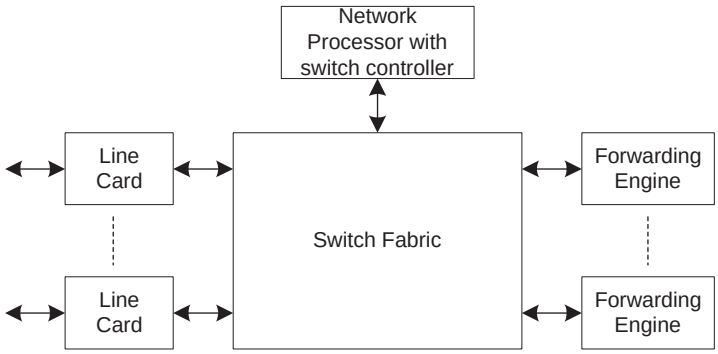

Fig. 3. General structure of high-speed router [10]

there may be fast processors and fast memories to process any incoming packet at link rates, an inefficient switching fabric may not sustain the aggregate throughput of on-board processors and memories. Consequently system throughput may not be as same as the link rate.

First generation router was built around conventional computer architectures with a shared backplane bus, a CPU, shared pool of memory and some line cards connected to the media. Packets arriving at any line cards are transferred to the CPU through a shared bus. All processing and forwarding decisions are made in the CPU and buffered in a central shared memory until the outbound link becomes free. Finally the packets are transferred through the shared bus to outbound line card/s and transmitted to the media connected to the next hop. Though this design is attractive for its simplicity, it has the disadvantage of having data crossing the shared bus several times, imposing a severe system bottleneck.

Fast processors with cache memory in the line cards and in the forwarding modules can reduce some dependence on shared bus. This approach was taken in designing second generation routers. However, bus based architectures always have a traffic dependant throughput as there is always a physical limitation on bus speed. Besides only one port is given permission to use the bus at a time.

Third generation routers (Fig 3) were designed with switch fabric architectures instead of a shared bus to alleviate this bottleneck. Here a switch fabric is used for non-blocking interconnection of time critical components with much higher capacity and speed than a traditional backplane bus [10]. Most common switch fabrics have been discussed elaborately in [8], [11] and [9]. These are (a) shared medium switch fabric, (b) shared memory switch fabric, (c) distributed output buffered switch fabric and (d) space division or crossbar switch fabric with input buffers.

Shared medium switch fabric is like bus based backplanes of first generation routers. Due to the physical limitation of bandwidth capacity, a shared medium

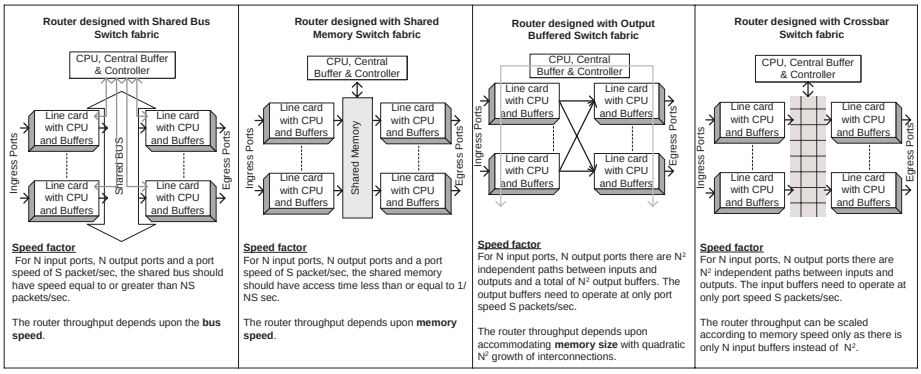

Fig. 4. Comparison among various switch fabrics

switch fabric imposes a serious performance bottleneck for inter-module traffic flow.

Shared memory switch fabric connects input and output ports to a central memory pool in parallel. As a result, input and output ports can have simultaneous read and write accesses to a shared memory. This can provide better throughput than a shared bus architecture. But this approach is limited by memory access time.

An output buffered switch fabric architecture can improve this limitation by splitting the shared memory into separate output buffers for each output port. There exists a mesh to connect all input ports to their respective output buffers. This approach creates scalability issues for backplane layout and memory size in systems with large port counts. On the other hand, an input buffered crossbar switch fabric is sometimes more attractive solution for its highly scalable, low cost and non-blocking switching solution.

A crossbar switch fabric is an alternative to alleviate aforementioned limitations. With some improvements in crossbar architecture, it can even achieve a terabit throughput rate [12]. A crossbar is formed by connecting all input and output ports in such a manner that inter-port traffic flows in unicast fashion. All input ports have their own buffer. So, the speed of memory buffer need not to be more than that of its associated port.

2.3 Working Modules

Various design approaches have been proposed by many researchers to achieve router throughput close to the link data rate. Commonly known designs are described and compared in [8], [9] and [10]. Many leading high-speed router manufactures, such as CISCO, 3Com, Lucent Technology and NetStar, have provided their routing solutions based on these basic principles. Like them, we have also decided to follow the generic router design approach (Fig 3) for our SAHN routing module.

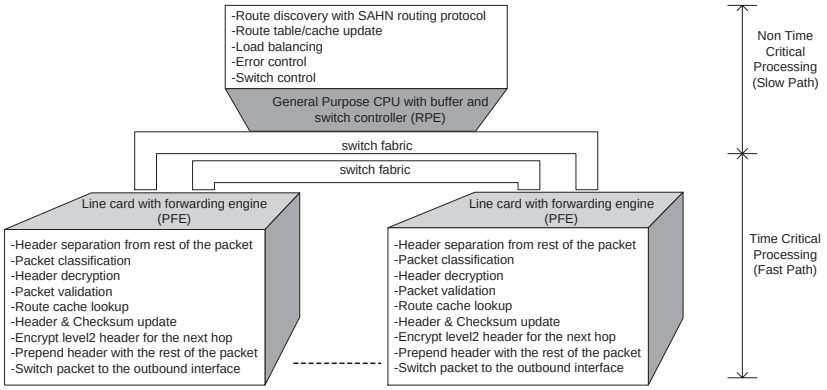

Fig. 5. Time critical and non-time critical functional modules in SAHN router

CISCO has its own forwarding engine called NPE (network processing engine) which combines line cards on a midplane. The backplane uses a crossbar switch fabric to interconnect the midplanes. The NetStar's GigaRouter includes the forwarding engine within the line cards [13]. The main idea is to reduce the flow of inter module traffic over the switch fabric. However in [14], a gigabit router design has been proposed with separate line cards, forwarding engines and the central controller unit, connected through a switched backplane. It has been argued to have forwarding engines distinct from the line cards for two reasons. One reason was they were not sure if they had enough board real estate to fit both the forwarding engine functionality with the line card functionality on a target card size. Another reason was to have more scalability. They found many router designs where the line cards were built with inadequate processors, throttling the performance to the processor's speed. To keep up with modern state of the art technology, they thought it was better to have separate forwarding engine which could be upgraded with the fastest processors if required. The last argument is one of our design goals too. But keeping the forwarding engine apart from the line cards may introduce a performance bottleneck in the switch throughput as the switch fabric will be allowing more inter module traffic flows.

Depending on various approaches available, the SAHN routing module follows a hybrid approach. There is a separate forwarding engine called the packet processing engine (PPE) connected to each line card and a central routing processing engine or RPE to perform the non-time critical tasks. The PPE, connected with its line card, forms the packet forwarding engine or PFE. The PFEs are connected to each other through a suitable switch fabric. Rather than using the same switch fabric, a separate switch fabric is used to connect the RPE with the PFEs. Using a distinct RPE to PFE connection is very important for the SAHN. The traditional routers are used mostly in structured infrastructure rather than a dynamic ad-hoc environment. Thus route discovery is not that frequent in ad-hoc networking infrastructure. Sharing the same switch fabric for

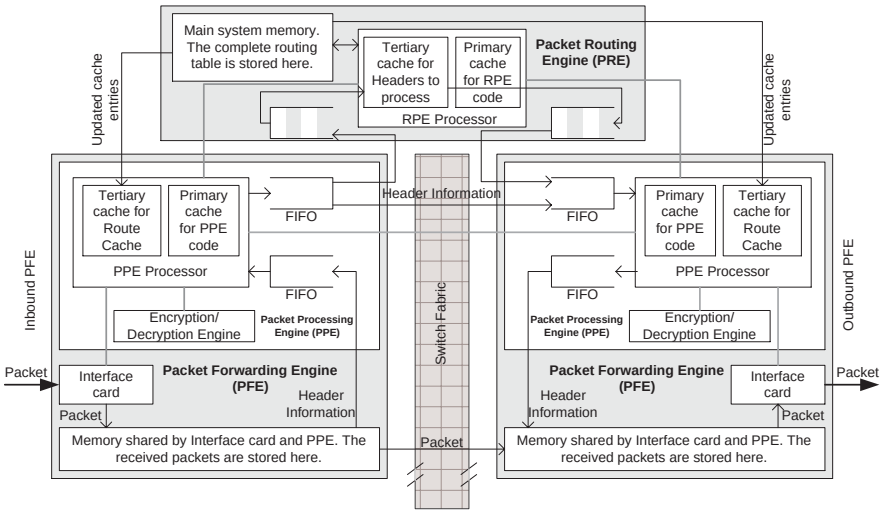

Fig. 6. A possible design outline of the SAHN routing module

both time critical and non-time critical tasks may lead to severe traffic congestion and poor system throughput. Whether to use a crossbar switched fabric or traditional bus based switch fabric for the non-time critical path, has to be tested in a real environment.

A question may arise whether this hybrid design framework is enough for the SAHN routing module. The answer is very straight forward. The forwarding engine in [14] used a DEC Alpha 21164 processor and achieved an overall rate of 111 Mpps. Other design approaches discussed in this section with fast crossbar switching fabrics and forwarding engines (CISCO uses MIPS processors in their NPE) are used for large routers in the multigigabit/terabit range with more than 500 Kpps packet switching rate [13]. But for the SAHN we are considering only a few network interfaces with media speed around 50 Mbps. Then we are targeting packet switching rate in the range of 25 Kpps to 50 Kpps which can be easily achieved with the derived hybrid design framework. Fig 5 shows the functional partitioning of tasks with associated hardware modules for the SAHN router. Before we give details of the modules it should be mentioned that our design approach has been influenced greatly by the solutions of CISCO, 3Com, Lucent and Linksys. Actually the basic principles followed by the major companies are almost the same. So it is reasonable that our design framework will be a variant of these available approaches.

The Packet Forwarding Engine (PFE) This section outlines the design of our network processing engine. Each of the packet forwarding engines has a direct link between its PPE and the corresponding network interface card (see Fig 6). The PFE is responsible for the followings major stages: (a) packet receiving stage, (b) fast packet switching stage and (c) packet transmitting stage.

Fast packet switching is done if the next hop information to the destination is already in the PPE cache. If the route to the destination is not known or if the packet is one of the special control packets like 'Hello', 'Hello Reply', 'Route Request' (RREQ), 'Route Reply' (RREP) or 'Route Error' (RERR), the routing processing engine (RPE) is employed. We will discuss the RPE in next section.

The inbound interface card receives a packet by copying it from the link into the receive memory buffer. Soon after a packet is received, the associated NPE processor is notified by an interrupt signal. Each NPE shares the same memory buffer with its interface card so that packet is not copied from buffer to buffer. This approach saves inter buffer transfer latency. The level 1 header is checked for classification. If it is not one of the special control packets and not destined for this router, the level 2 header is separated, decrypted and validated. Considering only the header rather than the whole packet makes the processing faster with small amount of memory. Now the forwarding/switching cache is examined to determine if the next hop information for the destination exists. If the next hop information is not available in the cache, the PPE puts the header into the RPE's queue and generates an interrupt to the RPE processor to let it know a header is waiting in the input queue to be processed. Otherwise the packet header is updated with the information from the cache. Then the outbound PPE is determined to forward the packet.

The updated header is queued into the outbound PPE and notified by an interrupt signal. The outbound PPE updates some of the header fields like the next hop MAC address, the HTL (hop to live), the TT (transmission time) etc. Then the level 2 header is encrypted according to the negotiated encryption scheme with the next hop and prepended with the rest of the packet. The QoS module in the PPE takes the remaining responsibilities. The QoS module provides flow control which is a part of load balancing. Depending upon the value of RTT (round trip time), packet length and destination, the QoS module schedules the packet's flow by putting it into the appropriate position in the input queue of the outbound interface. If there is congestion in the queue, it is the responsibility of the QoS module to discard the packet. Finally the packet in the front of the queue is transmitted through the outbound interface card and an interrupt signal of successful transmission is sent to RPE processor to update some values in the routing table.

The Routing Protocol Engine (RPE) The non-time critical tasks are handled by the routing protocol engine or RPE. These mainly include tasks of (a) managing routing tables, (b) updating forwarding caches and (c) handling the control packets (Hello, Hello Reply, RREQ, RREP and RERR). Update in the forwarding caches for load balancing is also performed here periodically. The RPE consists of a general purpose processor, and its own memory modules to hold the routing table.

Once the RPE processor receives the interrupt signal from a PPE, it assumes that a header has been inserted in its input queue. If it's any of the control packets, the whole packet is also transferred from the PFE memory buffer to the

RPE memory area. In all cases the header in the RPE's input queue contains a pointer to the memory location of the packet (whether it is in the PPE memory or in the RPE memory). An appropriate process is determined and scheduled to process the header in the front of the queue.

If the packet is not destined for the router itself then, after it has been processed, the appropriate outbound PFE is determined by consulting the central route table. The header is updated with the relevant next hop information and passed to the input queue of the outbound PFE with an interrupt signal to its PPE.

If the header corresponds to any of the control packets, the central routing table is updated with the new route information. The RPE periodically invalidates the aged entries in the central routing table and downloads the updated next hop information into the forwarding caches. The main advantage of having separate forwarding caches and routing table is that the forwarding caches only have to indicate the next hop entries for a particular destination. For this reason, the forwarding tables are much smaller and can be maintained in the cache of the PPE processor [14]. Decoupling the processing of route and next hop updates from the fast processing of the forwarding engines make them to work independently at a higher throughput. All the fast routers at present follow almost the same technique.

3 Operating System for the Routing Module

Previous sections had presented discussions about the hardware architecture for the SAHN routing protocol. In this section we will discuss to select an appropriate realtime operating system (RTOS) as a base for implementing the routing protocol. The RTOS must be able to execute the routing protocol properly and give us the option to port the final product into an embedded system.

Before going any further we may state the reason behind using a RTOS instead of a normal OS. The SAHN routing module is a real-time system since the time critical tasks have to be performed in real time. So, it is reasonable to use an appropriate real-time operating system (RTOS) to run the routing module. The RTOS for our purpose, should have the following properties.

- The higher-priority tasks must always be executed in preference to the lower-priority tasks. There should be support for fixed-priority preemptive scheduling for all of its tasks. Interrupt latency as well as the amount of time required to perform a context switch should be as small as possible. These are important because they represent overheads across the entire system.
- It must be as cheap as possible. Sometimes the RTOS source code is necessary to resolve problems with the application code.
- It has to be highly portable to various processor families as faster processors will continue to emerge. This makes the system more scalable to upcoming technologies. So, an application code written with the present RTOS can be used as a standardized piece of code for future projects.

	VxWorks Wind River System www.windriver.com	**ThreadX** Green Hills Software www.ghs.com	**C Executive** JMI Software Systems, Inc.	**QNX Neutrino** QNX Software Systems Ltd. www.qnx.com	**RTLinux** FSMLabs www.fsmlabs.com	**LynxOS** LynuxWorks www.lynx.com	**Embedix** Lineo www.lineo.com	**RTAI** DIAPM RTAI www.aero.polimi.it/~rtai/ (many supports are provided by Lineo)
Target Supported	PowerPC, Coldfire, 68K, Intel Architecture, Intel StrongARM and XScale, ARM, SuperH, MIPS	PowerPC, Coldfire, 68K; MCORE, ARM7, ARM9, ARM/Thumb, StrongARM, SH, TriCore, XScale, StarCore, ZSP, i960, V8xx, MIPS	PowerPC, 29K, 68K, ColdFire, i960, MIPS, SH, SPARC, V800, MIPS	PowerPC, x86, 175PowerPC0 A, MIPS	PowerPC, x86, Alpha6, MIPS	PowerPC, PowerQUICC, PowerQUICC II, Intel Architecture, MIPS	PowerPC, ColdFire, Dragonball, ARM 7 & 9, StrongARM, SuperH , x86/IA32, MIPS	PowerPC, x86 (with and without FPU and TSC), ARM (StrongARM; ARM7: clps711x-family, Cirrus Logic EP7xxx, CS89712), MIPS
Development Host	Self-Hosted	Self-Hosted	UNIX, Solaris, Windows	Self-Hosted, Linux, Solaris, Windows, QNX4	Linux	Solaris, SunOS, RS6000, LynxOS	Linux, Self-Hosted	Linux
Languages Supported	C, C++	C, C++	C, C++, Assembly	C, C++, Assembly, Java	C, C++	C,C++, Ada, Pascal, Java, Modula-2	C, C++	C, C++, PERL
Min ROM/RAM required (KB)	15/5	2/1	5/1	64/varies	1500/4000	37/11	10/10	2000/2000
Typical context switch time/Interrupt Latency(µs)	10	1.7(40MHz ARM7)/0.5(200MHz PowerPC)	3/2 (100MHz)	1.95/4.3(Pentium 133), 2.6/4.4(Pentium 100)	<30 (Interrupt Latency on 486/33MHz PC)	4 -19/14	7/15	4/20
Multitasking Strategy	Round-Robin, Time slice, Tasks can dynamically alter priorities, Rate monotonic Scheduling	Time slice, Fixed priority, Tasks can dynamically alter priorities	Time slice, Fixed priority, Tasks can dynamically alter priorities	Round-Robin, Time slice, Fixed Priority	One-shot, Periodic, FIFO, Rate monotonic	Round Robin, Time slice, Fixed priority, Tasks can dynamically alter priorities, Rate monotonic		One-shot, Periodic
Multiprocessor Support	Yes	No	No	Yes	Yes	Yes	Yes	Yes
Source Code Included	No	Yes	No	No	Yes*	No	Yes*	Yes*
Base price (USD)	$3000-$4000	$7500+ (Royalty Free)	$2500 (Royalty Free)	$3,995 (run time $50/ seat)	Free. (Royalty free).		$149. (Royalty Free)	Free*

*as per GNU Public Licensing agreement

Fig. 7. Comparison among various RTOSs

- It must be capable of supporting multiple processors simultaneously. This is important as there are some tasks in the PFE and the RPE modules that may require their individual processing power at the same time to achieve desired performance.
- Its image should be small enough to fit in a small ROM/Flash-disk as our end product will be embedded.
- It should have a familiar development environment, possibly POSIX compliant. Familiarity and competence with the development environment rather than struggling with a new working platform can effectively reduce the development time of any software product.

Taking the aforementioned properties as the quantitative measures, we have made comparisons among some of the well reputed RTOSs in the current market. These have been summarized in Fig 7. Readers are referred to [15], [16], [17], [18] for more details. Some relative comparisons have been presented graphically in Fig 8 from an analysis performed by University of Wisconsin.

It is apparent from these figures, the RTOSs providing more scalable properties (different types of processor support, less ROM/RAM requirements etc) tend to be far more expensive in terms of both upfront costs and recurring royalty/licensing fees. Some of them are not provided with source codes. So, initially we have decided to work with the RTOSs which are freely available like RTLinux or RTAI (They are free under GNU Public Licensing agreement). Their underlying Linux kernels can give the flexibility to develop and test our SAHN routing modules in any Linux system. Moreover there have been some real world tests

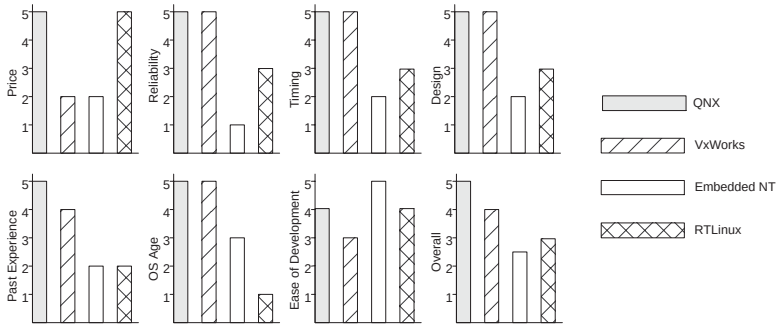

Fig. 8. Relative comparison among various RTOSs

and performance evaluations of ad-hoc routing protocols[19][20][21] in the Linux environment. All these experimental setups were based upon the Linux kernel for its familiarity, robustness, and more importantly, free availability. Though it is evident from Fig 7 that the Linux solutions require more memory, memory is getting cheaper day by day.

At the end here is a simple argument for our intention to use an of-the-shelf RTOS rather than building one of our own. Our routing module is separated into simple smaller modules (RPE, PPE etc). Each of them can be implemented in a smaller embedded system. In that case we may not need the complicated scheduling, context switching of any available RTOS. Each of the small modules can have their own lower abstraction layer written by us. But initially it will be time consuming to write and test our own RTOS. A better approach is to create the application layer (the SAHN routing module) and test it with an appropriate of-the-shelf RTOS. We will make the routing module as general as possible so that it can be ported to any RTOS for performance evaluation. In future it can be integrated with our own optimized RTOS.

4 Conclusion

In this paper we have proposed a possible design outline and implementation framework for an efficient on-demand routing protocol suitable for ad-hoc community networks, such as the SAHN. At present we are developing and testing our routing protocol in the GLOMOSIM (ver2.03). We have also built a testbed with desktop PCs to test our work in real environment. Each of these PCs acts as a SAHN node. Currently a node is capable of communicating with other two nodes through wireline networking technology. Eventually this wireline network will be replaced by wireless networking and each of the PCs by individual integrated routing modules as proposed in Section 2. Though we believe that more optimizations and changes may be required during prototype implementation and testing, the proposed architecture and survey analyses can be adapted to many ad-hoc community networks.

References

1. 24/01/2003. http://www.wirelessanarchy.com.
2. R. Pose and C. Kopp. Bypassing the home computing bottleneck: The suburban area network. *3rd Australasian Comp. Architecture Conf. (ACAC)*, pages 87–100, Feb 1998.
3. E. Makalic A. Bickerstaffe and S. Garic. CS honours theses, Monash Uni, 2001. www.csse.monash.edu.au/~rdp/SAN/.
4. D.B. Johnson. Routing in ad-hoc networks of mobile hosts. Technical report, IEEE Workshop on Mobile Computing systems and Applications, Dec. 1994.
5. Royer E.M. Perkins C.E. and Das S.R. Adhoc on demand distance vector (AODV) routing. IETF Internet Draft, Nov. 2000. http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-07.txt.
6. R. Pose M. M. Islam and C. Kopp. Efficient Routing in Suburban Ad-Hoc Networks (SAHN). *The 2003 International Conference on Communications in Computing (CIC 2003)*, June 23-26 2003. In Press. Las Vegas, USA.
7. R. Pose M. M. Islam and C. Kopp. Routing In Suburban Ad-Hoc Networks. *The 2003 International Conference on Computer Science and its Applications(ICCSA 2003)*, July 1-2 2003 2003. In Press. San Diego , California , USA.
8. Aweya James. Ip router architectures: An overview. Nortel Networks. Ottawa, Canada, K1Y 4H7, 05/01/2003. http://www.owlnet.rice.edu/~elec696/papers/-aweya99.pdf.
9. A new architecture for switch and router design, 05/01/2003. http://www.pmc-sierra.com/pressRoom/pdf/lcs_wp.pdf.
10. G. Minshall P. Newman and L. Huston. Ip switching and gigabit routers. *IEEE Comms. Magazine*, Jan. 1997.
11. Sayrafian Kamran. Overview of switch fabric architectures, July 2002. http://www.zagrosnetworks.com.
12. N. Uzan S. Papavassiliou and J. Yang. The architecture design for a terabit ip switch router. *IEEE Workshop on High Performance Switching and Routing*, pages 358–362, 2001.
13. Gigabit networking: High-speed routing and switching, 05/01/2003. http://www.cis.ohio-state.edu/~jain/cis788-97/gigabit_nets/.
14. C. Partridge et al. A 50-gb/s ip router. *IEEE/ACM Transactions on Networking*, 6:237–248, June 1998.
15. Dedicated systems encyclopedia, 12/11/2002. http://www.realtimeinfo.be/encyc/-BuyersGuide/RTOS/Dir228.html.
16. Embedded systems programming, 12/01/2003. http://www.embedded.com/story/-OEG20021212S0061.
17. SDTimes software development, 13/01/2003. http://www.sdtimes.com/news/027/-emb4.htm.
18. i Appliance Web, 13/01/2003. http://www.iapplianceweb.com/appDirectory/-IAW_OPERATING_SYSTEMS.
19. S. Desilva and S.R. Das. Experimental evaluation of a wireless ad hoc network. *9th Int. Conf. on Computer Comms. and Networks (IC3N), Las Vegas*, Oct. 2000.
20. E.M. Royer and C.E. Perkins. An implementation study of the AODV routing protocol. *EEE Wireless Comms. and Networking Conf. (WCNC)*, 3:1003–1008, 2000.
21. S.J. Lee S.H. Bae and M. Gerla. Multicast protocol implementation and validation in an ad hoc network testbed. *IEEE Intl. Conf. on Comms. (ICC)*, 10:3196–3200, 2001.