Routing Technology Final Report (NCR-9318902)

Craig Labovitz, Masaki Hirabaru

Merit Network, Inc
4251 Plymouth Road
Ann Arbor, MI 48105-2785

1 Summary

Under the National Science Foundation Routing Technology Grant (NCR-9318902), Merit Network, Inc. successfully developed a suite of tools for the management and distribution of inter-domain policy-based routing information for the Internet. The toolset, named the Multi-Threaded Routing Toolkit (MRT), includes a stand-alone multi-protocol route server, routing protocol test traffic generators, and a number of routing protocol information visualization and analysis tools. The final release of the MRT toolset enjoyed an enthusiastic response from users in the research and vendor communities, as well as from Internet service providers. Several router vendors have adopted MRT simulation tools for regular use in their product development cycles. Network service providers, including Sprint, MCI and ANS, actively use MRT-based tools to monitor their networks and debug routing problems. The next-generation of the DARTNet testbed, CAIRN, has chosen to deploy the MRT routing daemon software throughout their network to supply IPv6 routing information.

The MRT project explored a number of novel approaches to routing architecture design and protocol implementation. Research topics included methods of parallelizing routing protocol computation, an evaluation of an object-oriented, library-based approach to routing protocol implementation, and the inclusion of recent advances in operating system technology into routing protocol implementations. The MRT tools make use of parallelized light-weight processes, multiple processor support, and shared memory, to gain improvements in speed, scaling and robustness over traditional routing architectures. The object-orient, library-based modular design of MRT encourages the rapid addition and prototyping of experimental routing protocol and inter-domain routing policy algorithms.

2 Architecture

2.1 Need for Parallelism

The exponential growth in the size and topological complexity of the Internet has placed severe strains on the backbone routing infrastructure. Ongoing network problems, including route flaps, limits on routing table memory, protocol implementation bugs, and configuration errors have led to failures of large segments of the Internet on a regular basis. Unless routing technology rapidly improves in scalability and reliability, the frequency and severity of network outages will continue to grow as the Internet expands.

Until recently, most modern routing platforms could be modeled as a single, monolithic process that handled both packet forwarding and routing protocol communication. In these architectures, the forwarding of packets and the distribution of network reachability information were tightly coupled. In recent years, router vendors, including Cisco, IBM and BayNetworks, have moved towards architectures that perform packet switching on separate, specialized hardware interface cards. In these architectures, a central CPU performing routing policy communication and policy calculation provides the interface cards with continually updated copies of the router forwarding table. The forwarding cards then cache all, or portions of the table, and switch packets independently of the central processor.

Although the current generation of commercial routers have gained significant improvements in switching speed by separating packet forwarding from policy computation, the routers still face a bottleneck in the protocol communication and policy calculation that must occur on the main

CPU. All but the most recent router models have experienced difficulties handling the level of policy computation and routing traffic exchanged at the major Internet Network Access Points (NAPs). The only current method of scaling routing policy computation and communication requires the addition of ever faster processors and more memory to routers. Traditional routing protocol technology does not scale over multiple processors. On multi-processor machines, a single-threaded routing software implementation only runs on a single processor at a time. As the trend towards hardware vendors releasing multi-processor architectures continues (e.g., SGI Challenge, SPARCCenter 2000, Dec AlphaServer), we wanted to develop a routing architecture that can distribute operations over multiple processors as well as over multiple, distributed machines.

In the MRT project, we address the scaling problems associated with routing policy communication and computation by incorporating multi-processor support and parallelized algorithms into our design. The use of multi-processor technology and kernel thread support in routing daemon software offers a number of benefits, including: speedup of route processing and policy calculation, improved scaling, avoidance of the task starvation problem, simplification of routing software architecture, fault tolerance, and improved support for real-time protocols.

Most recent routing protocol specifications, including IDRP and BGP4, were designed with the use of multiple threads of execution in mind. BGP4 and IDRP describe several independently scheduled phases of route processing. One phase performs routing policy calculations, another phase conducts route selection, and the third phase updates the routing information base (RIB) and advertises the newly chosen routes to peer routers. The standards documents (RFC-1654 and ISO-10747) further note that later phases must block while earlier phases complete their processing. A threads-based routing protocol implementation allows for a simpler, more natural architecture for managing these phases and their associated asynchronous routing events. In general, the highly asynchronous nature of routing protocol processing can be modeled better by a program control flow of multiple concurrent actions than by a finite automaton model.

Of course, the implementation of modern routing protocols does not require threads. Several routing implementations, including the GateD routing daemon, successfully implement multiple protocols using only a single kernel thread of program control. These implementations usually maintain several priority event queues which schedule both routing protocol I/O and a variety of periodic operations, such as scanning network interface cards and monitoring kernel routing table changes.

The major difficulty encountered with the application, or user-level management and scheduling of routing protocol events involves the issues of task starvation and fairness. In GateD, for example, most tasks are uninterruptible. Once GateD begins a task, such as removing all unreachable entries from the routing table, the task runs to completion. A time intensive task, such as establishing a low bandwidth BGP peer with a full routing table, forces all other tasks to block until it completes. This blocking creates significant starvation problems for protocols with time-dependent requirements, such as the 30 second Keep-Alive timer in IDRP and BGP. During times of high routing instability or processing activity (such as during GateD reconfiguration and periods of network media problems), GateD occasionally drops valid BGP peering sessions due to timer expiration.

This is not to say that the problems of starvation and fairness in single threaded routing implementations are insurmountable. Several protocol implementations, including GateD, have recently added limited support for task preemption and swapping. These application-level implementations of task preemption require much of the same call stack and priority queue machinery as used by operating systems for the scheduling and swapping of processes. But unlike the operating system, these user-level implementations lack the kernel's immediate knowledge of machine resources and process requirements. Without this knowledge, the user-level implementations of swapping are necessarily less efficient than the kernel in switching multiple threads of program execution. In general, the effective user-level implementation of threads adds a significant layer of code complexity to routing software and provides only a subset of the functionality offered by kernel level threads.

Heavy weight threads, or traditional UNIX processes, provide an alternative method for distributing routing computation over multiple processors. In this method, each routing protocol instance might be implemented as an individual heavy-weight process which communicates with other protocol processes and the kernel via some form of inter-process communication (IPC). Heavy-weight processes, however, are less efficient and more costly than kernel level threads in managing multiple routing protocols. Unlike threads which share data space, different processes cannot easily share common data structures. Instead, heavy-weight processes must communicate by using relatively resource expensive system supplied mechanisms, such as pipes, and shared memory. The most significant resource penalties come with the expense of the CPU swapping processes. In addition to the storing and copying of call stacks and registers, process swapping often also requires the switching of page tables, signal vectors and kernel stacks.

2.2 Parallelism in MRT

2.2.1 Overview

Using kernel-level threads, the MRT architecture recreates the complex task and job queue functionality of single-threaded routing software implementations in a relatively straight-forward manner. Instead of re-implementing the kernels' process swapping code, the MRT architecture makes extensive use of kernel thread services through the standard POSIX threads application programming interface (API). The POSIX threads interface provides for transparent scheduling, execution, and swapping of multiple process threads of control. The POSIX threads implementations also include provisions for advanced features, such as the setting of thread run priority, exclusion locks and the maintenance of private thread data space. Overall, we believe that the kernel, with its immediate knowledge of system resources and I/O, is better suited to handle scheduling than a user level process.

The main impetus for using threads and multi-processor technology in routing software implementations is speedup. On multi-processor systems, multiple threads can run on multiple processors concurrently. Threads also can effectively mask the latency in I/O operations and take full advantage of their CPU time slice.

In designing threaded network applications, researchers have taken three different categories of approaches to problem of achieving parallelism with threads: packet, functional and layered parallelism. In packet-level parallelism, a separate thread is associated with each arriving and out-bound packet. Applying this approach to the TCP/IP stack, a separate thread shepherds each IP packet from the network to the application layer performing the appropriate processing at each stage. In layered parallelism, a separate thread is associated with each layer of the network stack. As an individual network layer completes processing on a packet, or series of packets, the thread passes the processed information on to a thread higher or lower in the stack. Functional parallelism associates threads with specific tasks, or functional units of processing. Examples of functional processing include associating a thread with ICMP processing.

2.2.2 Event Parallelism

The original design of MRT used a variant of packet parallelism that we termed event-parallelism. In this model, individual threads were associated both with the processing of individual I/O events, such as the receipt of protocol packets, and regularly scheduled protocol events, such as the periodic advertisement of RIP routes. An event-parallelism model seemed an intuitive, straight-forward approach to the problem of applying parallelism to routing protocol processing. The model also followed closely on the successes of similar approaches in TCP/IP packet parallel processing.

But early in the MRT software development, we concluded that an event-parallelism model was not well-suited to routing protocol implementations. An early profiling of our code revealed that a disproportionate amount of time was spent handling mutex locks and semaphores. This dependency on locks and semaphores arose out of two factors: the sequential scheduling requirements of protocol processing, and the large number of shared resources required by most routing protocols. In BGP, for example, the processing of each in-bound protocol packet requires access to a number of common variables and memory locations, including shared-buffer space and a protocol state machine table. Each BGP thread was required to lock and unlock every shared resource during processing. Profiling of our code further showed that event threads were frequently forced to block while waiting to obtain mutex locks for resources currently controlled by other threads.

The most significant problem with our event-parallelism model involved the sequential processing requirements of routing protocols. Protocols events, such as the advertisement and subsequent withdrawal of a prefix, require deterministic, ordered processing. Maintaining the correct order of route announcements and subsequent withdrawals is critical for achieving the correct end-state of a network topology. Since thread events are independently scheduled by the kernel, the MRT software had to resort to locks and queues to enforce a particular order on the processing of packets. Beyond the relatively high overhead of managing these locks, the scheduling requirements also introduced a significant amount of artificial latency. Achieving the correct order of protocol packet processing required that threads wait for confirmation of the completion of earlier scheduled threads. This latency effectively countered any speedup gained by the benefits of having multiple parallel threads of execution. Overall, associating threads with protocol events resulted in a large number of short-lived processing elements contending for a large number of shared resources. The number of locks and critical sections of code added a prohibitive level of software complexity and increased the probability of deadlocks and other code synchronization problems.

2.2.3 Functional Parallelism

In an effort to achieve better performance, limit the contention between threads for shared resources, and simplify our software, we altered the MRT architecture to more closely model a functional model of parallelism. In the new MRT architecture, we associate a thread with each routing protocol functional entity. In practice, this means that one or more threads is associated with each routing protocol instance (BGP, RIP, RIPNG), and with each routing processing element, such as the routing information base (RIB) and the I/O controller. Each functional entity and associated thread are modeled as objects which have calling methods and maintain control of their own private resources. Objects communicate with other objects through specific call routines, or methods. Threads now only require locks or semaphores when they access a resource controlled by another object, such as BGP interacting with the RIB. In the majority of processing, objects do not require locks. For example, the BGP thread handles packet decoding, updating the state machine, and policy computation independently of other threads. BGP only needs locks at the final stage of processing when it updates the address family RIB. Overall, the coarser grain of parallelism introduced by this functional approach requires a smaller number of threads, fewer mutex locks, and helps limits the contention between threads for shared resources.

Each MRT object maintains a FIFO queue of pending events, or tasks. Events are scheduled by other objects when they call one of an object's methods. For example, a timer thread might call BGP.timerfire() to schedule BGP timer processing. When an object's event queue is empty, the object blocks and waits for a conditional signal from another thread indicating that a new event has been scheduled. All objects include methods for clearing and resorting their event queues. External manipulation of event queues is needed in a number of protocol situations, including the transition of a BGP peering session from established to idle state. In this case, all packet announcement events must be abandoned to maintain correct functioning of the protocol.

A minority of MRT threads do not maintain event queues. These threads, which we call services, only schedule events with other objects and do not conduct any event scheduling or processing of their own. Examples of service threads include a timer and I/O monitor thread.

The current implementation of MRT includes the following threads: timer, select, BGP, RIP, RIPng, user-interface, and a main, controlling thread. Since most UNIX implementations only associate a single timer per process, the MRT timer thread is used to multiplex the single process timer over multiple threads. Objects like BGP and RIP schedule alarms by calling the timer scheduler method and providing both an interval or an absolute time, and a call-back method. The timer thread maintains a sorted queue of time-based events which associate pending alarms with objects and their call-back methods. The timer service also includes mechanisms for one-time alarms and adding jitter to timer intervals.

The select service performs synchronous I/O multiplexing on behalf of MRT objects. Since objects already block while waiting for the scheduling of new events, the objects require another mechanism for learning of pending I/O. The select thread monitors object socket descriptors for pending read, write and exception events. Objects register an interest in socket by calling a select service method. Once the select service detects an I/O event, the service invokes the call-back method of an object and stops monitoring the socket. After an object completes processing of a socket, it notifies the select thread to once again begin monitoring.

Overall, the new MRT architecture has proven simple to implement, easy to debug and moderately resource efficient. The software still requires a significant number of mutex locks, but not as large a number as that required by our earlier architecture. Although we currently lack quantitative results measuring relative performance of MRT and other routing daemons, subjective results suggests that threads have provided substantial improved performance on multi-processor machines.

2.3 Memory

Memory management has proven to be the single largest hurdle in designing the MRT architecture. The memory problems arise from the difficulty in sharing memory between threads. Because threads are scheduled independently by the kernel, MRT can never guarantee the order in which threads will access memory. Specifically, when one thread allocates memory and passes a pointer to another thread, the first thread requires some mechanism for learning that the second thread has released the memory before it can deallocate the memory. For example, a BGP object thread needs a method of discovering that the RIB object no longer requires access to the shared BGP route advertisement pointers. Solutions to the thread memory management problem include: copy memory on passing, garbage collection, and never deallocating memory. Although trivial to implement, the third option of never deallocating memory is not especially efficient. The MRT design implements a combination of memory copying and garbage collection.

For long-lived, frequently referenced, and large bundles of memory, such as BGP and RIP2 attribute blocks, MRT maintains a memory reference count for each attribute block and performs garbage collection. On the receipt of an in-bound packet, protocol threads "check" the attribute block into a central library. The attribute library either returns a pointer to an already allocated, similar memory block, or adds the memory block to the library and returns the same pointer. Every "check-out" of a library attribute block increments the reference count on the memory. When thread objects complete processing and release an attribute block, the attribute block is checked back into the library and the reference count is decremented. When the reference count reaches zero, the memory is released from the master library and deallocated.

In the case of smaller, shorter-lived, and less frequently referenced memory blocks, MRT threads copy memory on passing to other threads. For example, the BGP object copies its linked-list of withdrawn BGP prefixes when calling the RIB's routing table change method. Since the memory is copied, both the BGP and RIB object are free to alter and delete the memory blocks. In addition to simplifying the code complexity, the copy on passing method also avoids the overhead associated with garbage collection and the maintenance of reference counts. Overall, copying is an inexpensive way of handling smaller chunks of memory, such as a 4 byte IPv4 prefix.

With the exception of prefix and attribute memory, MRT objects control their own memory and resources. The objects are otherwise free to allocate and deallocate memory at will.

3 Libraries

The ability to rapidly prototype new protocol languages and routing policies is critical to the evolution of Internet technology. As the pace of routing innovation continues to quicken, early implementations provide an invaluable guide to theoretical limitations and potential implementation hurdles. Examples of emerging technologies include IPv6, RSVP, Multicast, BGP communities, and the Routing Policy System Language (RPSL).

The modular, object-oriented design of the MRT libraries encourages rapid prototyping and experimentation. Instead of a monolithic software architecture, MRT is based on a number of independent libraries offering a variety of routing protocol services. The libraries include an interface to kernel routing tables, a user telnet interface, a Cisco configuration library, a patricia tree routing database, support for multiple event timers, and libraries for the construction of routing protocol packets. Following the traditional Unix model, researchers may select only the libraries they need and may interchange and create new libraries as required.

The MRT project has used these libraries to explore route server technology, protocol simulation and analysis tools, and next-generation routing protocols, such IPv6 RIPng.

The MRT libraries include a clearly defined application programming interface (API) and a manual with examples and tutorials on using the libraries.

4 Deliverables

The MRT project researched and developed a number of tools for the management, analysis and distribution of inter-domain policy-based routing information. The software tools have proved a valuable asset to router vendors, researchers and network service providers. The current toolset played an important role in the smooth transition from the NSFNet backbone to the new NSF Internet architecture. Network service providers, including Sprint, MCI and ANS, are actively using MRT-based tools to monitor their networks and debug routing problems. The toolset includes software for generating and analyzing route flap statistics, programs to generate real-time graphical maps of Internet routing, and tools to compare real-time routing information against information registered in the Internet Routing Registry (IRR).

MRT tools have contributed to a better understanding of what is needed to administer, manage and operate large inter-domain networks. Researchers using the MRT routing stability tools have uncovered serious BGP routing pathologies and router vendor bugs.

The complete MRT toolkit is available at http://www.merit.edu/ mrt. The MRT work has been discussed at NANOG, IETF, and IEPG meetings. In addition, announcements about the availability of the toolkit have been sent to several Internet-wide newsgroups and mailing lists.

The MRT tools fall into four main categories:

1. routing daemon
2. record and replaying of network routing traffic
3. synthetic workload generation
4. characterization of routing information

4.1 Routing Daemon

The MRT project developed a UNIX-based, hardware independent multi-protocol, Multi-threaded Route Server daemon called MRT. The MRT daemon includes route server support for multiple views, and the BGP4 and RIP routing protocols. The software provides kernel-level support for a variety of hardware and software architectures, including NetStar, SunOS, BSDI, FreeBSD and Solaris.

MRT supports a subset of the Cisco-style policy and configuration language. The MRT daemon also provides an interactive, Cisco-like telnet configuration and monitoring interface.

The CAIRN research community has chosen to deploy the MRT routing daemon on their wide-area testbed to supply IPv6 routing information to their network.

4.2 Recording and Replaying of Network Routing Traffic

The MRT distribution includes several simulation tools that provide for the logging and replaying of all protocol traffic seen in any routing exchange, such as all of the BGP traffic exchanged at a Network Access Point. The tools, named SBGP and SRIP, read log files of recorded routing packet events to build and later inject copies of the routing packets directly onto a network media. The tools support the replaying of packets at the same, real-time rate as they were recorded.

These simulation tools provide a deterministic, reproducible means of evaluating routing software and hardware performance under identical routing traffic loads. The tools provide router vendors and Internet service providers with the ability to exactly reproduce "real-world" routing problems and scenarios. For example, an ISP can evaluate how well a new router handles the volume of BGP routing updates exchanged at Mae-EAST, or a researcher can repeatedly replay AADS BGP NAP data to evaluate the performance of new route flap dampening algorithms.

4.3 Synthetic Workload Generation

This category of MRT simulation tools provides a means of generating synthetic routing protocol traffic based on textual or graphical abstract description language. The simulation tool, named BGPSim, allows the specification of a variety of network characteristics, including the number of routers, the topological connectivity (autonymous system paths), and path selection. By describing various router parameters, new protocols can be evaluated under a variety of network environments. Several router vendors, including NetStar and GateD, have adopted BGPSim for regular use in their product development cycles.

Lacking workload generation tools like BGPSim, it is difficult, if not impossible to recreate "real-world" routing conditions using a local test bed. Even a large test bed (100+ routers) cannot generate the volume of routing instability and topological diversity currently seen at the Internet core. The BGPSim tool provide a means of generating traffic conditions exceeding worst-case scenarios seen to date at the Internet core. For example, the tool can generate instability exceeding 100+ BGP prefixes/second, routing tables with 100,000+ prefixes, and corrupted/non-compliant routing packets. Using the BGPSim tool, protocol and network developers can explore the scaling properties of routing protocols and routers.

4.4 Characterization of Routing Information

Operational networks and test-beds have an ongoing need for tools to characterize routing environments and detect potential routing problems. The MRT project developed a number of tools to monitor routing stability, provide real-time access to routing tables, and compare live routing information against registered policies. The MRT and Routing Arbiter research efforts took advantage of the synergy between the two projects to cooperatively develop a number of the tools described in this section.

The MRT tool RouteGen queries the Route Servers at each NAP to provide real-time information on the routes installed in each view. RouteGen provides both a World Wide Web GUI interface into the routing data, as well as a set of raw data files available for anonymous FTP.

The MRT tools BGPCheck and CiscoCheck compare the consistency of real-time routing traffic with policy preferences registered in the IRR. BGPCheck monitors BGP information exchanged between routers by peering with a router, and CiscoCheck compares the output of a Cisco routing table dump with information registered in the IRR. The Routing Arbiter project uses these tools to compile a daily Web page of IRR and Internet routing discrepancies.

The MRT project also developed a number of tools that analyze routing traffic patterns to provide automated detection of routing information and routing stability/performance problems. This class of MRT tools provides network characterization and analysis by examining the logs of routing protocol packets recorded by previously described MRT tools. Examples of the statistics and analysis provide by these tools includes: the origin and distribution of of route flap in a network, a topological map of autonymous system connectivity, and the relative stability of network links. Daily MRT-based reports on routing statistics and networks problems are available on the Routing Arbiter Web site at http://www.ra.net/statistics.

5 Other Work

We have reviewed the routing protocol simulation and research activities of a number of other groups. In all the cases described below, the MRT work explores novel avenues of research and tool development.

The University of New Hampshire Routing Consortium has proposed the development of a routing protocol network monitor, Graphical Routing Analysis System (GRASS). As described on their Web page, the tool provides for the decoding and logging of intra-domain routing protocol packets. The tools do not provide the MRT functions of editing, automatically creating, and later replaying/injecting of routing protocol packets. The tools also appear limited to intra-domain routing protocols (RIP, OSPF). The distribution of the tools is also restricted to UNH IP Consortium members.

Several other researchers have developed self-contained network simulators, including MARS developed at University of Maryland, NS developed at Lawrence Berkeley National Laboratory Labs, and NEST from Columbia University. The MARS tool is designed to test protocol correctness and generate TCP/IP packets and flows. The tool does not generate routing protocol traffic.

Scott Bradner at Harvard University has developed tools and methodologies for testing network hardware performance and interoperability. Bradner's test suites concentrate on router/bridge performance in the switching of packets, and not on routing protocol communication.

6 Project Notes

The National Science foundation awarded Merit Network, Inc. the Routing Technology grant, NCR-9318902, effective May 15, 1994 through April 30, 1995. In this grant, Merit proposed a subcontract with Cornell Theory Center to provide a publicly available routing daemon supporting multiple policy images.

The deliverables of the project were to be:

1. a modified GateD which represents active routes on a per-peer basis in its routing information base (RIB)
2. a specification for a language to represent inter-domain routing requirements
3. a tool for checking the consistency of traffic paths with path preferences
4. a tool to query one or more of the routing images to check which routes are installed in the routing table for each peer session

In the first year of the project, Cornell began the process of transitioning the GateD Consortium to another host institution. Unfortunately, the GateD Consortium was unable to complete the necessary modifications to the GateD software before the transition began. Concerned that a major piece of the proposal would remain incomplete, Merit decided to explore a new architecture for accomplishing the goals of providing a new platform for routing protocols and routing engines. This project lead to the development of the Multi-Threaded Routing Toolkit (MRT).

In April 1995, Merit received a one year no-cost extension to the Routing Technology project to complete work on the multi-threaded routing software.

7 Bibliography

References

[1] BGPD mailing list, bgpd-request@merit.edu

[2] S. Bradner, A. Mankin. IPng Internet Protocol Next Generation. Addison-Wesley, 1996.

[3] B. Catanzaro, *Multiprocessor System Architectures.* Prentice Hall PTR, 1995.

[4] Classless Interdomain Routing Discussion Mailing List, ciderd-request@merit.edu

[5] B. Chinoy, "Dynamics of Internet Routing Information," Proceedings of SIGCOMM '03, pp. 45-52, September 1993.

[6] M. Cekleov et. al. "SPARCCenter 2000: Multiprocessing for the 90's!", Proceedings of IEE COMPCON, pages 345-353. San Francisco CA, February 1993.

[7] S. Dawson and F. Jahanian, "Deterministic Fault Injection of Distributed Systems," in *Lecture Notes in Computer Science: Unifying Theory and Practice of Distributed Computing.* Springer-Verlag, September 1994.

[8] S. Dawson and F. Jahanian, "Probing and fault injection of protocol implementations," in *Proc. Int. Conf. on Distributed Computer Systems*, pp. 351–359, Vancouver, B.C., May 1995.

[9] S. Dawson, F. Jahanian, T. Mitton, "A Software Fault Injection Tool on Real-Time Mach". Proceedings of the IEEE Real-Time Systems Symposium, Dec. 1995

[10] D. Estrin, Y. Rekhter, S. Hotz, "Scalable Interdomain Routing Architecture," Proceedings of SIGCOMM '92, pp. 40-52. August 1992.

[11] M. Goldberg, G. Neufeld, M. Ito. "A Parallel Approach to OSI Connection-oriented Protocols," Third IFIP International Workshop on Protocols for High-Speed Networks, pages 219-232, May 1993.

[12] C. Huitema, *Routing in the Internet*, Prentice Hall PTR, 1995.

[13] S. Kleiman, D. Shah, B, Smalders. *Programming with Threads.* Prentice Hall PT, 1996.

[14] C. Labovitz, "A Distributed, Scalable, Multi-threaded Network Routing Architecture". Under development.

[15] B. Lindgren, B. Krupczak, M. Ammar, K. Schwan, "An Architecture and Tool-kit for Parallel and Configurable Protocols," Proceedings of the International Conference on Network Protocols. San Francisco, CA, October 1993.

[16] The POSIX Threads standard, IEEE Std. 1003.1c-1995.

[17] C. Labovitz, MRT Home Page, http://www.merit.edu/m̃rt

[18] GateD Manual, http://www.gated.org

[19] Mikler, A., Wong, J., and Honavar, V. (1995). An Object-Oriented Approach to Modeling and Simulation of Routing in Large Communication Networks. Under review. Draft available as: ISU CS-TR 95-09.

[20] E. Nahum, D.Yates, J. Kurose, and D. Towsley, "Performance Issues in Parallelizing Network Protocols", Proceedings of the First Symposium on Operating System Design and Implementation, Usenix Association, Nov 1994

[21] North American Network Operators (NANOG) Mailing List, nanog-request@merit.edu

[22] V.Paxson, "End-to-End Routing Behavior in the Internet". Under Review. 1996.

[23] J. Salehi, J. Kurose, D. Towsley, The Effectiveness of Affinity-Based Scheduling in Multiprocessor Networking". Under Review.

[24] D. Schmidt, R. Suda, "The Service Configurator Framework". Proceedings of the IEE Second International Workshop on Configurable Distributed Systems, Pittsburgh, PA, March 1994.

[25] J. Yates, E. Nahum, J. Kurose, D. Towsley, "Networking Support for Large Scale Multiprocessor Servers". Technical Report CMPSCI95-83. University of Massachusetts.

[26] Y. Rekhter and P. Gross, "Application of the Border Gateway Protocol 4(BGP-4)," RFC 1711, DDN Network Information Center, March 1995.

Contents

1 Summary **2**

2 Architecture **2**

2.1 Need for Parallelism 2

2.2 Parallelism in MRT 4

2.2.1 Overview 4

2.2.2 Event Parallelism 5

2.2.3 Functional Parallelism 5

2.3 Memory 6

3 Libraries **7**

4 Deliverables **8**

4.1 Routing Daemon 8

4.2 Recording and Replaying of Network Routing Traffic 8

4.3 Synthetic Workload Generation 9

4.4 Characterization of Routing Information 9

5 Other Work **10**

6 Project Notes **10**

7 Bibliography **12**