Cache Management of Dynamic Source Routing for Fault Tolerance in Mobile Ad Hoc Networks

Ching-Hua Chuan and Sy-Yen Kuo

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan

Email: sykuo@cc.ee.ntu.edu.tw

Abstract

Mobile ad hoc networks have gained more and more research attentions by provisions of wireless communications without location limitations and pre-built fixed infrastructure. Because of the absence of any static support structure, ad hoc networks are prone to link failure. This has become the most serious cause of throughput degradations when using TCP over ad hoc networks. Some researches chose Dynamic Source Routing (DSR) as the routing protocol and showed that disabling the assigning of a route directly from cache gives better performance. In this paper, we introduce an efficient cache management mechanism to increase the TCP throughput by replying with a route directly from the cache of DSR and perform the cache recovery when a host failure has occurred. We use simulations to compare the performance of our algorithm with the original DSR under the link failure prone environment due to mobility. We also provide the simulation results when host failures are considered in the ad hoc networks.

1. Introduction

An ad hoc network is a dynamic network consisted of a group of mobile devices which communicate with each other by wireless media. Communications only can be done when a node is in the wireless transmission region of another node. Through a group of intermediate nodes willing to forward packets, a source can send data to a destination which is not in its communication region. In an ad hoc network, every node may work as a host or a router at some time.

There have been a lot of routing protocols proposed for ad hoc networks [6-11]. The Dynamic Source Routing (DSR) protocol [7] is an on-demand routing protocol based on the concept of source routing. The protocol consists of two major phases: route discovery and route maintenance. Whenever a source has a packet to send, it first checks its routing table to see if there is a route to the destination. If not found, then the source broadcasts a route request. When an intermediate node receives a route request, it again broadcasts this request by appending its address to the request packet until this packet reaches the destination. The destination replies to the first arrived request. It sends a route reply to the source containing the whole route from source to destination. When this packet reaches the source, the connection is established and all the subsequent packets will go through the route with the route in their packet headers. To reduce the number of route discoveries, each node maintains a route cache for the routes it has learned. The cache is updated by route error messages. If the protocol allows reply to route request from cache, the intermediate nodes can send route reply to the source with the route it keeps for that destination and stop broadcasting the route request.

Because of TCP's inability to recognize the difference between link failure and congestion, the link breakage due to mobility makes the TCP throughput lower. Simulation was used in [2] to measure the TCP performance while choosing DSR as the routing protocol. In [2] the authors also indicated that the better TCP performance achieved with no reply from cache is due to the fact that there are no stale routes returned. Because DSR has no mechanism to immediately respond to dynamic network topology changes, more routes may be stale when the mobility becomes high. Under such circumstances, reply from cache introduces more routing errors.

It is necessary to find an efficient cache management strategy for DSR. Since DSR is an on-demand source routing protocol, sending link layer beacons periodically for getting signal strengths disobeys the protocol's design idea. In this paper, we introduce a new cache management

0-7695-1414-6/01 $17.00 © 2001 IEEE

mechanism that incurs no other protocol overhead and can be easily added to the original DSR. We also extend the use of route cache to perform the cache recovery from a host failure.

The paper is organized as follow. Section 2 presents our algorithm of cache management and demonstrates it under two circumstances, link failure and host failure. In section 3, the simulation methodology and simulation environment are described. Section 4 gives simulation results and analysis. Finally section 5 provides a conclusion.

2. Protocol Description

In this section, we present a detailed description of our cache management protocol. The goal of the cache management is to avoid replying with a stale route from the cache. We try as much as possible to keep the correct routes in the cache and reflect the dynamic network changes by a local mechanism. First we state the main idea and definition of the cache mechanism. Then we describe how the protocol works in a more detailed way. Finally we go through some examples under two failure situations - link failure and host failure. One note must be taken first, which is that all the broadcasted control messages are sent only to the neighbors.

2.1. Protocol overview

Since the problem of replying from the cache is due to the probability of returning stale routes, the first thing we have to do is to identify whether a route in the cache is correct or not. When a node receives a route request, it checks its cache and returns the route which is identified as correct. By doing so, we can still use the cache to reduce the number of route requests and incur no throughput degradation due to reply error.

In this paper, we use a mark to prevent incorrect reply. A route marked *"stale"* is a route which is likely broken. When receiving a route request, we can't return routes marked "stale". Note that while a route marked as "stale" can not be used for route reply, it can still be used to send packets. We only determine the neighbor links' stability by monitoring the signal strength of received packets from the neighbor node. If we found some neighbor link is likely broken, we mark all the routes that go through it as "stale". We define a period for each "stale" route to indicate how long it should be kept in the cache. If we get some information showing that a "stale" link works again before it timeouts, then we recover it as a normal one. Otherwise, discard the expired stale routes. The values of stale route's period and signal strength threshold are adjusted dynamically depending on the network condition.

Before we recover a "stale" route, we must make sure it is correct. So we define two control messages: *"confirm"* message and *"route ok"* message to achieve this goal. These messages are initiated by a stale route which could be recovered as normal. By using these two messages we also can do the cache recovery after a host failure. When discarding a "stale" route, the node checks whether it has another substitute path in its cache. If not, it broadcasts a *"link broken"* message to its neighbors to inform them of the link error. Each node receiving a "link broken" message discards the error link and then does the same thing to find a substitute route. If not found, then it broadcasts the message again. When a node receives a "link broken" message and finds that the broken link does not exist in the cache, then it discards the message and does nothing.

We assume that a host can realize its failure after it restarts again. When a host restarts, it broadcasts a *"host needs recovery"* message to its neighbors. When receiving a "host needs recovery" message, a node should work the same as the signal strength is beyond the threshold again. So the nodes that received the "host needs recovery" message send back a "confirm" message with routes. After checking that the next node in the route is alive, the route can be stored in the cache of the host who needs recovery.

In the following we describe our protocol in details and explain how they work.

2.2. Signal strength

A node uses the signal strength of the received packets to determine the stability of neighbor nodes who pass (or send) the packets to it. Each node in an ad hoc network only cares about the stability of neighbor links. We define a signal strength threshold as a parameter to determine a neighbor's condition. When a received packet's signal strength is under the threshold, we consider this packet's sender to be unstable. When a node realizes that there is a possibility to lose some of its neighbors, it marks all the routes that go through that node as stale. We let each node only monitor its neighbors' stability to reduce network and protocol overhead. We mark all the route as unstable to ensure that no error route could be returned from the route cache. The value of signal strength threshold can be adjusted according to mobility patterns or dynamically adjusted based on the statistical results of network topology changes. In our implementation we give different values to the threshold for different node moving speeds.

All the actions of our mechanism are driven by signal strength determination. When a neighbor's signal strength first goes below the threshold, we mark all the routes passing through it as stale. Within the stale route's duration, if there is no matching "confirm" message to the marked route received, then discard the stale route from the cache. When a neighbor's signal strength first goes beyond the threshold, we send "confirm" message to that node with the route marked as stale. The signal strength determination pseudo-code is given in Table 2.1.

Table 2.1. Pseudo-code for signal strength determination

```
/* Signal Strength Determination */
if (SS>=Threshold)
 {  if ( packet sender does not belong to the neighbors )
      {  Add it into the neighbors.
         if ( a stale route belongs to packet sender )
          { Send confirm message to it. }
      }
 }

if (SS<Threshold)
 {  if ( packet sender is one of the neighbors)
      {  Discard it from the neighbors.
         if ( a route with packet sender as next node )
         {  Mark the route as stale.
            Record the marking time.
         }
      }
 }
```

2.3. "*Confirm*" message

The "confirm" message is used to get validation before recovering a "stale" route as normal. If a neighbor is sensed to be alive again and there are some "stale" routes belonging to it, then there is an opportunity to recover the routes. Since the signal strength determination only guarantees that the first link of the path is active now. Therefore before recovering a "stale" route, the node must check the status of the route with its neighbors. Look at Fig. 2.1 to consider the situation. Node A senses B's signal strength beyond the threshold for the first time and wants to recover the "stale" route BCX as normal. Node A sends a "confirm" message to node B to ask whether the route BCX can work or not. The thing that node A wants to know is whether it can send packets to B if it has packets with destination C or X. Therefore the thing that node B has to provide is the guarantee to answer A's request.

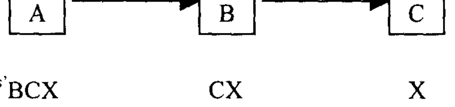

Fig. 2.1. Node A sends confirm message to node B to validate route.

2.4. "*Route ok*" message

This message is used to respond to the "confirm" message. When a node receives a "confirm" message and has an active match to the route, it responds with the whole route to confirm the route which is to be recovered. The same as the "confirm" message, the "route ok" message is one per route. Use the same example in Fig. 2.1, when B receives A's "confirm" message and if B has route CX in its cache, then B sends "route ok" message to A with route BCX. If B doesn't have any match with A's "confirm" message or the route CX is marked as stale, then B won't do anything. Node A can recover the route BCX as normal only when B's "route ok" message is received.

2.5. "*Link broken*" message

We keep "stale" routes in the cache for a while. When the stale route's duration expired and no matching "route ok" message is received, then the route will be discarded. When a node discards a route from its cache, it broadcasts "link broken" messages to its neighbors (TTL=1). The message contains a node id, which is the node that the message sender has a problem of connecting to. When a node receives a "link broken" message, it looks up its cache to find out if there is any route passing through the message sender and reaching the node included in the message. If so, then it discards the route and tries to find a substitute route to that destination (node included in the "link broken" message). If found, then it stops and does nothing further. If not found, then it broadcasts "link message" again with the same node id. If a node receives a "link broken" message and finds that it does not have any route to that destination, it can ignore the message.

Look at Fig. 2.2 for example. If C discards a "stale" route X, then it broadcasts a "link broken" message including node X to show that the node can not be connected. Both node B and node D receive C's "link broken" message. Because D doesn't have any route to destination X, D discards the message. Node B discards the route CX and finds that it has no substitute routes for destination X. So B broadcasts "link broken" message with node X again. Then both node A and node C receive B's "link broken" message. Because C just discards the route X and has no route to it, C discards the message and does nothing. Node A discards the route BCX and finds that it has a substitute route WX to X, so it stops and the process of discarding a stale route is finished.

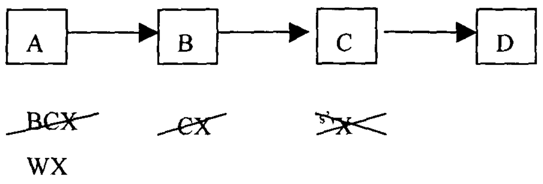

Fig. 2.2. An example for link broken message.

2.6. "*Host needs recovery*" message

This message is designed for the cache recovery from host failure. When a node restarts after its failure, it broadcasts "host needs recovery" message to its neighbors to inform them it needs some route information for recovery.

2.7. "*Neighbor*" message

When receiving a "host needs recovery" message, a node uses "neighbor" message to show that it is his neighbor now. This message is used only in the host failure situation for validation of the route in a "confirm" message. We use Fig. 2.3 to show the use of a "neighbor" message. After a short failure due to power shortage or battery change, node B restarts again and it loses all the routes in its cache. B broadcasts "host needs recovery" messages to its neighbors and both A and C receive the messages. Because A has a route through B, A sends a "confirm" message to B with the route BCX. On the other hand, C has no routes through B and therefore C sends a "neighbor" message to B. For A's "confirm" route BCX, because B got C's "neighbor" message which proves that B can connect with C, B can store the route BCX in its cache.

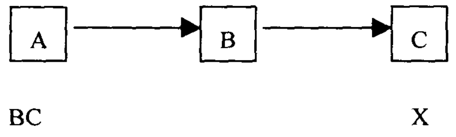

Fig. 2.3. An example to explain "neighbor" message's function.

2.8. Advantages and disadvantages

Here we discuss the advantages and disadvantages of our protocol. Our design goal is to use a simple method which is also compatible to DSR to avoid replying with stale routes. We did not add any overhead and just assigned a mark to a route that could be stale. Since the route marked as stale can still be used to send packets and cause no error, the TCP throughput can be increased. Besides, we only monitor the neighbor links' stability and exchange route information with neighbor nodes, these local mechanisms will not increase too much overhead to the network. We don't send any periodic information or signals so that we can keep the advantages of the on-demand DSR protocol. Finally our protocol is designed for fault tolerance. We can detect link failure in advance to keep the returned route correct and perform the cache recovery from a host failure.

The main drawback of our protocol is the method we determine a link's stability. We passively wait for packets and measure their signal strength and use timers to refresh when we get no packets. Because it can not exactly reflect the situation about the links, it may cause some unnecessary actions. For example, a link may be stable but we discard it since no packets are received from it. Another is the way we judge a link to be unstable. It is better to use a more accurate method to do it.

2.9. Route state

Each route has two states. The first is the normal state and can be used for routing and reply. The second is the stale state which can be used for sending packets but can not be used for reply. The transition between these two states is shown in Fig.2.4.

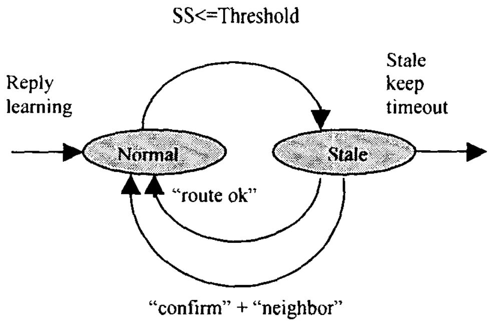

Fig. 2.4. Each route has two states.

3. Simulation Environment and Methodology

The results in this paper are based on simulations using the *ns* network simulator from Lawrence Berkeley National Laboratory (LBNL) [13], with extensions from the MONARCH project at Carnegie Mellon [4]. At the physical layer, the extended *ns* employs a radio propagation model supporting propagation delay, omni-directional antennas, and a shared media network interface. At the link layer the IEEE 802.11 Medium Access Protocol is provided. All results are simulated on a network configuration consisting of TCP-Reno over IP on an 802.11 wireless network, with routing provided by the Dynamic Source Routing (DSR) protocol and BSD's ARP protocol.

Our network model consists of 30 nodes in a 1500x300 meter flat rectangular area. The nodes move according to *random waypoint mobility* model. In this model, each node picks a random destination and speed in the area and then travels to the destination in a straight line. After it reaches the destination, it pauses and then picks another destination and speed to move again. All nodes communicate with identical, half-duplex wireless radios that are modeled after the 802.11-based WaveLan wireless radios, which have a bandwidth of 2Mbps and a nominal transmission radius of 250m. TCP packet size is 1460 bytes and the maximum window is eight packets.

The simulations are run for mean speeds of 1, 5, 10 and 25 m/s and pause times of 0 and 10 seconds for a period of 150 seconds. We use CMU's traffic and scenario generating scripts to create network traffic and mobility patterns. Each result is the average of 30 patterns. First we use only a single TCP traffic to simulate under link failure and host failure situations. Then we simulate all the patterns in the same way in multiple data connections. The multiple data connections consist of one TCP traffic and ten CBR connections across eight nodes, which send 512 byte packets at a mean rate of 5 packets per second. We use TCP throughput as the performance metric for the comparison of cache management strategies.

4. Simulation Results and Discussions

4.1. Link Failure

In Fig. 4.1, 4.2, 4.3 and 4.4, we show the variation of TCP Reno Throughput versus mean speed of nodes for a pause time of 0 and 10 seconds respectively. We use single TCP traffic in Fig 4.1 and 4.2 and multiple traffic sources in Fig 4.3 and 4.4. As can be seen, the throughput decreases with increasing mean speed of nodes. High mobility results in the increase of link failure frequency and TCP regards it as network congestion. So it reduces its window size and causes the throughput degrade.

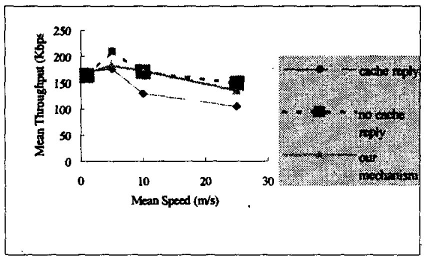

Fig. 4.1. Signal TCP traffic with pause time = 0 seconds.

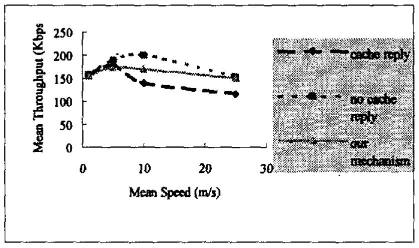

Fig. 4.2. Signal TCP traffic with pause time = 10 seconds.

Generally, our protocol has good performance in ad hoc networks under link failure circumstances. At high mean speed, our method performs better than the original DSR with reply from the cache. We solve the problem of replying with stale routes in the cache and get the results almost equivalent to disallowing route reply from the cache. Since the ad hoc network in our simulation is not big, route discovery doesn't need too much time. At high mean speed it causes less overhead for route discovery with no cache reply strategy than for handling link errors with the cache reply strategy.

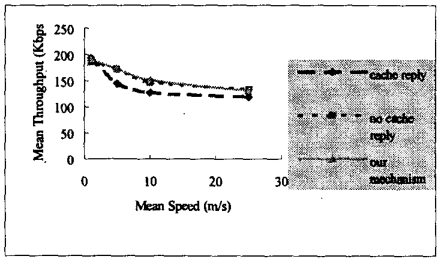

Fig. 4.3. Multiple traffics with pause time = 0 seconds.

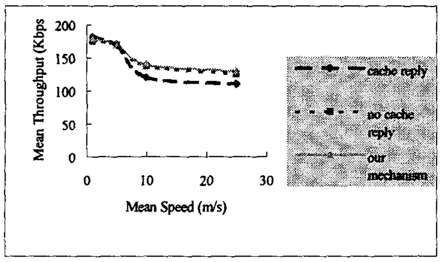

Fig. 4.4. Multiple traffics with pause time = 10 seconds.

4.2. Host Failure

In Fig. 4.5, 4.6, 4.7 and 4.8 we did the same experiment but add host failures in the simulation. We chose a node randomly as the failed host and crash its cache. We only simulated our protocol and DSR with cache reply in this part since crashing a cache doesn't have much meaning for no cache reply strategy. In the host failure simulations, our protocol has better results than the original DSR with reply from the cache. This means our method successfully recovers some crashed caches from host failures.

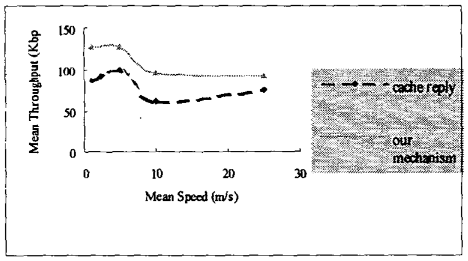

Fig. 4.5.Host failure with single TCP traffic and pause time = 0 seconds.

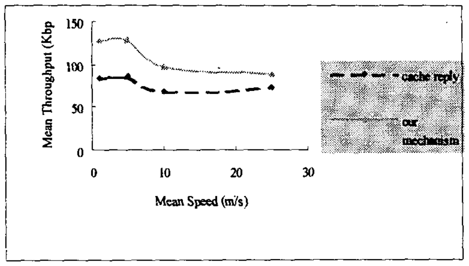

Fig. 4.6.Host failure with single TCP traffic and pause time = 10 seconds.

However, compared to the results in link failure environment, our protocol is still affected by host failures (The throughput in link failure environment is higher than host failure environment). We infer the reasons as followings. Although we try to recover a cache after it fails, the throughput still degrades due to the host failure. When a host fails and it loses all the routes in its cache, the packets which have been received also have to be discarded because there is no route to transmit them. In fact when a host fails, it loses not only the information in the cache but also all the data in the buffer.

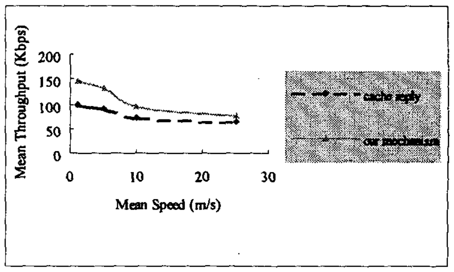

Fig. 4.7. Host failure with multiple traffics and pause time = 0 seconds.

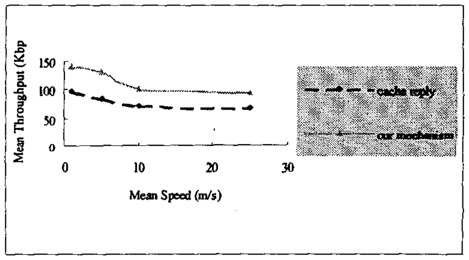

Fig. 4.8. Host failure with multiple traffics and pause time = 10 seconds.

5. Conclusions

In this paper we has shown that route cache management in an on-demand routing protocol has significant performance implications for TCP. We proposed a new protocol to solve the stale route problem with replying a route from the cache. We performed extensive simulations under single TCP traffic and multiple traffic situations to compare the results with other protocols. The simulations showed that we have improved the performance of reply from the cache in DSR significantly, which is almost equivalent to that of no cache reply mechanism. We have also considered host failure situations in the environment and discussed how they affect the TCP throughput. Besides using the same simulation methodology as link failure situations, we added host failures in the simulation environment. From the simulation results we observed that host failures also make the throughput much smaller and meanwhile we showed the benefit of our mechanism for the cache recovery. For future work we suggest that when designing a routing protocol for ad hoc networks, it is important to consider the host failures as well as the link failures. Also for fault tolerance other aspects recovery can be considered, such as the recovery of packets which have been received at a failed host, in addition to routing information recovery.

6. Acknowledgment

This research was supported in part by the Development of Communication Software Core Technology project of Institute for Information Industry and sponsored by MOEA,R.O.C.

7. References

[1] E. M. Royer, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks," IEEE Personal Communications, Vol. 62, April 1999, pp. 46-55.

[2] G. Holland and N. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks," in Proceedings of IEEE Mobicom'99, Seattle, WA, August1999, pp.219-230.

[3] G. Holland and N. Vaidya, "Impact of Routing and Link Layers in TCP Performance in Mobile Ad Hoc Networks," Wireless Communications and Networking Conference, 1999. WCNC. 1999 IEEE, 1999, Vol. 3, pp. 1323 -1327.

[4] J. Broch, D. A. Maltz, D.B. Johnson, Y. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols," in ACM/IEEE *Int. Conf. On Mobile Computing and Networking*, Oct1997, pp. 85-97.

[5] A. Ahuja, S. Agarwal, J.p. Singh, R.Shorey, "Performance of TCP over different routing ptorocols in Mobile Ad-Hoc Networks," in IEEE VTC 2000, Tokyo, Japan, May 2000, pp.2315-2319.

[6] R. Dube, C. D. Rais, K.-Y. Wang and S. K. Tripathi, "Signal Stability based Adaptive Routing (SSA) for Ad Hoc Networks," IEEE Personal Communications, Februbuary 1997, pp. 36-45.

[7] D. Johnson, D.A. Maltz, and J. Broch, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (Internet-Draft)," *Mobile Ad-hoc Network(MANET) Working Group*, IETF, Mar. 1998.

[8] Z. J. Haas and M.R. Pearlman, "The Zone Routing Protocol (ZRP) for Ad Hoc Networks (Internet-Draft)," *Mobile Ad-hoc Network (MANET) Working Group*, IETF, Aug. 1998.

[9] Y.-B. Ko and N. H. Vaidya, "Location-Aided Routing (LAR) in Mobile Ad Hoc Networks," in ACM/IEEE *Int. Conf. On Mobile Computing and Networking (MobiCom'98)*, October 1998.

[10] C. E. Perkins and E.M. Royer, "Ad Hoc On Demand Distance Vector (AODV) Routing (Internet-Draft)," *Mobile Ad-hoc Network (MANET) Working Group*, IETF, Aug. 1998.

[11] R. Sinakumar, P. Sinha, and V. Bharghavan, "Core Extraction Distributed Ad Hoc Routing (CEDAR) Specification (Internet-Draft)," *Mobile Ad-hoc Network(MANET) Working Group*, IETF, OCT. 1998.

[12] S. Agarwal, A. Ahuja, J. P. Singh and R. Shorey, "Route-Lifetime Assessment Based Routing (RARB) Protocol for Mobile Ad-Hoc Networks," Communications, 2000. ICC 2000. 2000 IEEE International Conference on Vol. 3, 2000, pp. 1697 -1701.

[13] K. Fall and K. Varadhan, ns *Notes and Documentation.* LBNL, August 1998. http://www.isi.edu/nsnam/ns/ns-documentation.html