Chapter 3

Methodology

We describe in this chapter our basic research methodology and key elements of our research infrastructure. We have used a combination of analysis, simulation, and experiments with real networks and protocol implementations to perform the research reported herein, and in the first section we describe our overall research strategy. Next, we describe the simulation environment used for our simulation studies and summarize the key extensions we have added. Finally, we describe elements of the Bay Area Research Wireless Access Network (BARWAN), which we used for our experimental work. We defer detailed descriptions of our measurement techniques and performance metrics to the later chapters.

3.1 Research Strategy

Our research strategy is depicted in Figure 3.1.[1] The figure indicates that we iterate cycles of analysis, simulation, and experimentation to converge on an effective solution to the research problems. The first phase of work was the definition of the problem and identification of performance bottlenecks. We started with two general problem areas critical to networking over next-generation broadband satellite systems: addressing the poor performance of the TCP protocol in a heterogeneous end-to-end environment that includes satellite channels (*satellite transport protocols*), and designing a core packet routing strategy for Low-Earth-Orbiting (LEO) satellite networks (*satellite routing*). In the case of satellite transport protocols, we first examined existing work in the field and used simulation and experiments with standard TCP implementations to uncover the causes of poor TCP performance over satellite channels. In this phase, we also relied on experimental results to validate our simulator, since we already had a working reference implementation. For satellite routing, we did not have access to any existing simulation tools or working systems, so our work in this phase was confined to examining the existing research literature.

Once we had a good idea of what the research challenges were, the second phase of work involved exploration of the design space and evaluation of potential solutions. Again, Figure 3.1 is a good illustration of the approach. In the case of satellite transport protocols, we first used the results of our benchmark performance results to analyze the problems and to formulate candidate solutions. Next, we used simulation to evaluate many of these solutions. In this phase, simulation was an easier

[1] This strategy was commonly used and cited by members of the BARWAN research team.

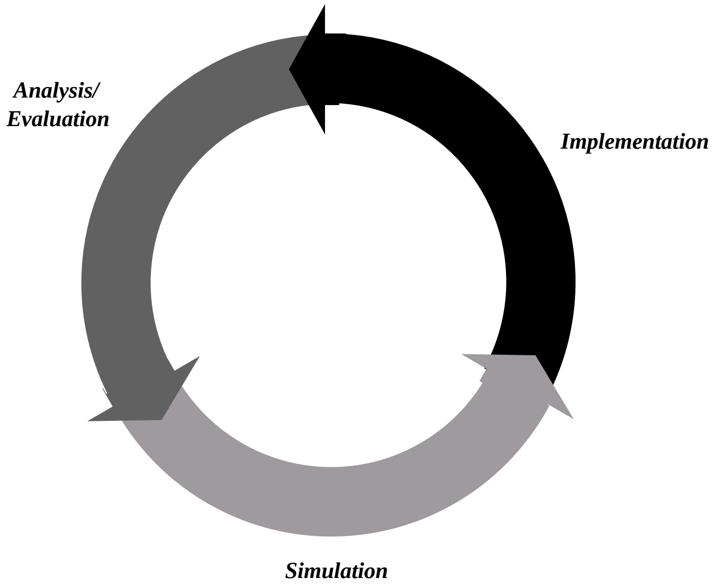

Figure 3.1: Three-phase research methodology– analysis, simulation, and implementation.

and more flexible approach to explore the design space than direct implementation was, because simulation is a controlled environment in which certain aspects of the design can be isolated and compared on an even basis. We also constructed a LEO network simulator by adding extensions to the *ns* simulator described below. Our extensions first used the existing routing infrastructure of *ns*; later, we improved the speed of simulation by optimizing the routing code for our network models. We then explored possible changes to satellite routing by constructing our own routing protocols and simulating their performance.

The third phase of the research consisted of iterative cycles of the second phase, refining our solutions as necessary until we were satisfied with the demonstrated improvements. As depicted in Figure 3.1, our simulation and implementation results led to further analysis and validation of improved solutions. The results of our research described in subsequent chapters were often the result of several iterations of the process.

3.2 Simulation Environment

Simulation is a particularly useful tool for networking research. First, it facilitates easy implementation of new algorithms and policies, allowing more rapid evaluation of a design space. Simulation can help to identify promising solutions which can often then be more carefully verified in an implementation. Second, a simulated environment is a controlled environment. Because of this, one can construct simulations that isolate the effects of certain parameters and algorithms on the overall performance. Also, evaluation of aggregate network performance is made easier because all network elements are made available through one interface. This is particularly important when studying the impact of an algorithm or policy on many nodes in a wide-area network, for example.

Third, in some cases, building an experimental implementation (such as a LEO satellite network or other large scale systems) is infeasible.

We performed most of our simulation studies using the UCB/LBNL network simulator known as *ns*, now widely used as part of the VINT project [8]. *ns* is a event-driven simulator originally derived from the REAL network simulator [72]. The simulator has an object-oriented architecture, and simulation objects are typically implemented as *split objects*: partly in C++, and partly in MIT's Object Tcl (OTcl) [141]. Such objects exist simultaneously in both language realms, and functionality can typically be added in either language (generally, functionality that requires per-packet processing is best implemented in C++, while more infrequently processed code is more flexibly implemented in OTcl). The state between the split implementation is made consistent through the use of *bound* instance variables, in which any changes to such variables in one language are immediately visible in the other. *ns* is a particularly strong choice for TCP research, since many TCP variants (Tahoe, Reno, NewReno, Vegas, etc.) are standard parts of the simulator. *ns* has also been used extensively for multicast routing and transport protocol research. Until recently, *ns* did not focus on providing detailed simulations of the link and physical layers, but UCB's BARWAN and CMU's Monarch research groups have contributed support for Local Area Networks (LANs), wireless channel error models, and wireless ad-hoc routing protocols. [20, 69].

Although we defer some details of our simulation extensions to later chapters, we briefly describe three enhancements we made to *ns*: (i) HTTP traffic generator, (ii) implementation of the Satellite Transport Protocol (STP), and (iii) LEO satellite network extensions.

3.2.1 HTTP Traffic Generator

TCP performance is well-known to be highly sensitive to the presence of other traffic in its path. In particular, the timing of packet losses due to congestion can cause the throughput to vary dramatically. When simulating TCP, it is necessary to load the foreground communications path with a realistic model of background traffic, so that performance can be accurately assessed in a realistic environment. We implemented, along with Emile Sahouria, an HTTP traffic generator for *ns*. This traffic generator was used to provide background Web-like traffic for both TCP and STP simulations, as described in the following chapters.

The HTTP traffic generator works as follows. Client and server traffic sources emulate the request and response traffic processes from typical Web browsers and servers. The client object first initiates a variable length request; after a random processing time, the server responds with a random number of connections of varying length. After a random viewing time (referred to as "think time"), the client then issues another request. Empirical distributions dictate all of the above random quantities. These distributions have been derived from traces of HTTP traffic taken by Bruce Mah on local area networks at the University of California, Berkeley, during the 1995-96 timeframe [79].

3.2.2 Satellite Transport Protocol

We implemented the data transfer mechanisms of the Satellite Transport Protocol (STP) in *ns* to test the large file transfer performance of the protocol. This simulation model then was used as a basis of the STP kernel implementation. We did not perform simulations of short-lived TCP or STP connections; instead, we relied on analysis and experiments with the actual implementations.

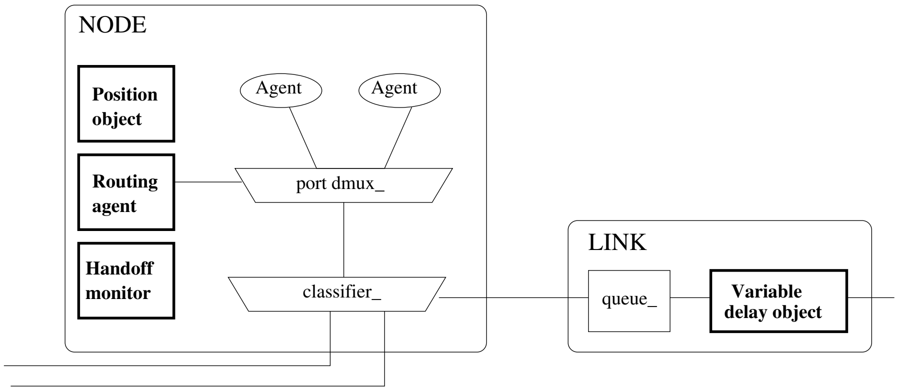

Figure 3.2: Extensions to the ns simulator (new elements listed in bold type).

3.2.3 LEO Satellite Network Extensions

As mentioned above, we selected *ns* as the basis for our simulation experiments because of its extensive support of Internet routing and transport protocols, and because our research group has been active in the ongoing development of the simulator. However, *ns* was not initially designed to support terminal mobility or dynamic topologies. Consequently, we were forced to introduce several new components into the simulator to more faithfully model LEO networks. Along the way, we attempted to make sure that our extensions were fully compatible with other aspects of the simulator, so that future researchers may investigate LEO networks along other lines (such as multiple access).

Figure 3.2 illustrates the major additions to the simulator.[2] We first introduced a spherical coordinate system, and added a *position object* to each network node. This position object can be given an initial coordinate and an equation which describes its trajectory through the coordinate system as a function of time. We centered the spherical coordinate system at the Earth's center, with the z-axis aligned with the Earth's rotation axis. This alignment simplified the description of polar orbits and trajectories for Earth terminals. The *link delay object*, which previously returned a fixed propagation delay, was changed to return a value based on the instantaneous positions of the two nodes at the end of the link.

The largest piece of coding involved link handoffs, because *ns* previously did not permit links to be dynamically detached and reattached to different nodes. Furthermore, we needed to introduce *handoff agents* to govern the handoffs. These agents are responsible for monitoring for opportunities to take down, bring up, or handoff links. Various policies for performing the handoffs can be implemented; we implemented asynchronous and synchronous handoffs as described above in Section 2.2.1. Finally, we implemented dynamic, distributed routing agents in each node for experiments with distributed routing described below in Chapter 6.

The default routing code in *ns* uses an all-pairs shortest path algorithm to compute new routes for each node in the simulator whenever the topology changes. This algorithm is useful to

[2]Since *ns* evolution is on-going, the exact structure of these enhancements that will be added to the publicly available simulator is subject to change.

populate routing tables initially for static topologies, but is very computationally expensive when applied to dynamically changing topologies because it has complexity of roughly $O(n^3)$. To speed up our simulations, we implemented single-source shortest path algorithms and configured the simulator to optionally compute routes on demand (whenever a packet needed to be sent), which yielded a run-time performance improvement of up to two orders of magnitude.

3.3 Experimental Testbed

We used the wireless testbed infrastructure of the BARWAN project at Berkeley. The BARWAN project was based on the vision of building future mobile information systems as a heterogeneous collection of *wireless overlay networks*. For example, a user may have a choice of connecting to an in-room infrared network, an in-building wireless LAN, a regional wide-area packet network, or even a satellite system. The research goals of this project were to tackle the problem of network access heterogeneity in such an environment. BARWAN solved problems associated with routing and handoffs within and between access networks, proxy-based application support, reliable transport over wireless channels, Web transport, and service location. References and a thorough overview of the project can be found in [20].

3.3.1 Experimental Machines and Software

Much of our experimental work involved machines on our local area networks, and most of the implementations involved changes to the networking code on the end hosts. We developed and experimented with modified TCP code and new Satellite Transport Protocol code on PCs running BSD/OS UNIX, version 3.0, from Berkeley Software Design, Inc. The networking stack in BSD/OS 3.0 resembles the code in the 4.4BSD-Lite distribution,[3] sometimes referred to as the "Net/3" release [127] and the source of many widely used systems like NetBSD and FreeBSD. This TCP/IP code has been developed and used over many years and is considered a stable source. Our experimental network consisted of 10 and 100 Mbit/s Ethernet segments joined by BSD/OS-based routers.

We used the `sock` program from Stevens to generate traffic for our experiments [127]. `sock` accesses the TCP/IP stack via standard system calls based on the well-known *sockets* Application Programming Interface (API) [127]. As described in later chapters, we sometimes used `sock` to simulate the transfer of large files, and at other times we used `sock` functions within another traffic generation program driven by traffic trace data. `sock` was trivially extended to support our STP experiments, as STP offers the same API as TCP, but via a system call with a different protocol number than TCP's.

We used the network trace tools `tcpdump` and `tracelook` quite frequently in our packet trace analysis. `tcpdump` was written by Jacobson, Leres, and McCanne; it uses the *BSD packet filter* [86] to put a network interface into promiscuous mode so that all traffic on the interface can be observed. `tracelook` is a Tcl/Tk program written by Greg Minshall for graphically viewing the output of a tcpdump TCP tracefile.

Some of our experiments involved emulating the transmission characteristics of satellite channels. Rather than use a sophisticated satellite channel emulator, we used modified Ethernet

[3] The 4.4BSD-Lite distribution refers to the April 1994 version of the source code for a common reference implementation of TCP/IP developed by the Computer Systems Research Group at the University of California, Berkeley.

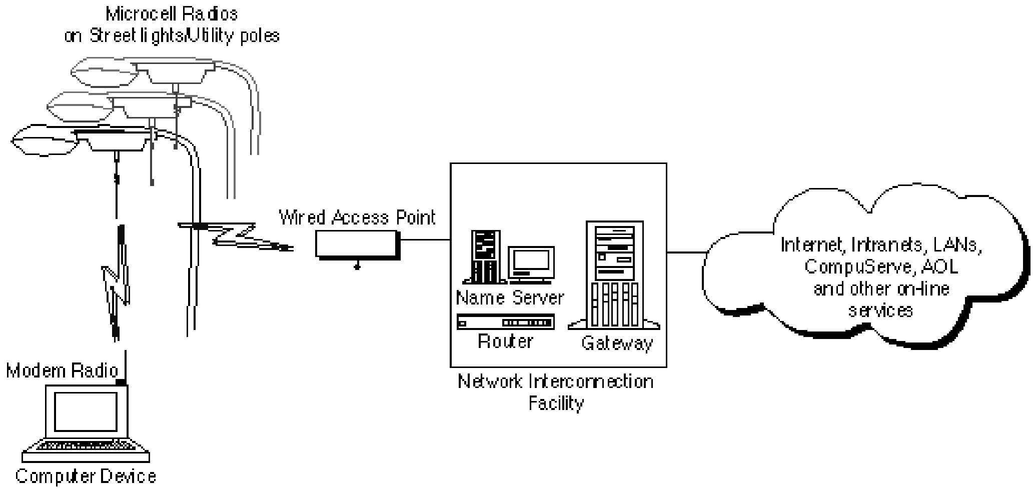

Figure 3.3: Architecture of the Richochet packet radio network (Source: Metricom, Inc.; used with permission).

device drivers (for BSD/OS) developed by Venkat Padmanabhan, a member of our research group. These drivers could buffer outgoing packets for a user-configured delay and also emulate a constrained bandwidth channel by imposing an additional delay based on the packet length and the emulated bit rate of the channel. We were also able to impose a random packet drop rate on the traffic through these drivers, which could be used to emulate a channel with a random, uniform bit error ratio. For transport protocol research over geostationary satellite channels, these drivers provided sufficient emulation, because the effects of a more precise modeling of the transmission characteristics of satellite channels are dwarfed by the dominant congestion-induced losses on Internet paths.

3.3.2 Ricochet Packet Radio Network

The Richochet packet radio network, deployed by Metricom, Inc., covers the Bay Area metropolitan region, as well as a number of other cities and airports in the United States. The system covers the region with telephone pole-top radios, and packets are routed through the radio network to one of several gateways to the Internet. The system uses frequency-hopping spread spectrum in the 915 MHz ISM band. The radios are half-duplex, meaning that they cannot simultaneously transmit and receive data. The radios also use a form of geographic routing to reach the nearest gateway; each radio is configured with its latitude and longitude, which it is able to announce to its neighbors, as well as the coordinates of a nearby gateway, and the radios route traffic to the neighboring radio that minimizes the geographic distance to the gateway. Figure 3.3 illustrates a modem that attaches to the serial port of a computer; the Point-to-Point protocol (PPP) [126] is used as an IP link layer between a computer and the gateway.

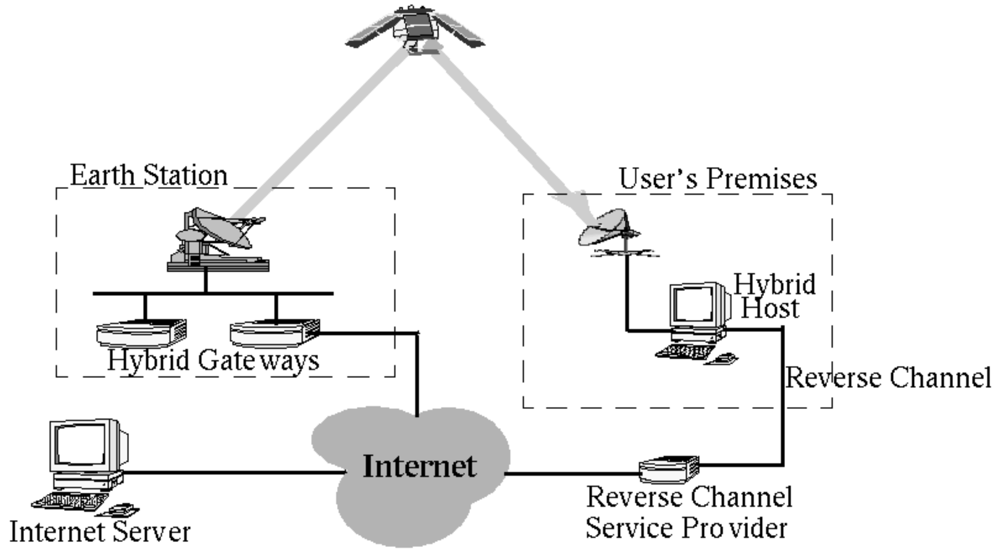

Figure 3.4: Architecture of the DirecPC satellite system (from [100]; used with permission).

3.3.3 DirecPC Satellite System

We used the DirecPC satellite system, developed and operated by Hughes Network Systems, for satellite experiments involving actual satellite channels. The DirecPC system is a hybrid Internet access system consisting of a high-speed unidirectional satellite broadcast channel and a return path accessed via a conventional Internet Service Provider (ISP). The system is based on the asymmetric traffic characteristics of typical end users, who use much more bandwidth inbound than outbound. Routing is performed by “tunneling” (encapsulating) outbound packets so that they are routed directly to the DirecPC network. From there, the packet is decapsulated and the inner, original, packet is sent onward to the destination Web site, but with a source address corresponding to the DirecPC network. In this manner, the packet can avoid being dropped by anti-spoofing filters commonly found in ISP networks, and the response traffic is naturally routed to the uplink gateway. Because of the asymmetric path, the end-to-end latency is around 400 ms, which is smaller than the 600 ms typically found on two-way geostationary satellite channels. Figure 3.4 provides an overview of the system.

To further our experiments, we placed a BSD/OS machine within the DirecPC network at their uplink facility in Germantown, MD. This enabled us to directly access the satellite uplink without traversing the Internet (which would have corrupted our forward data flow). We were able to remotely download our modified networking code to this machine.

3.4 Summary

In this chapter we have described our methodology and research infrastructure at a high level, while deferring detailed descriptions of experiments and simulations until later chapters when such details can be put in the proper context. We present the detailed results of our research in the next three chapters by first starting with the question of improving TCP performance over GEO satellite links.