Submission instructions: The project output, including software, electronic report, and hardcopy report, are due on Dec. 18, 2004 (5pm). The electronic material should be submitted using `turnin`*. The hardcopy can be slid under my office door (CS220). Other project related specifications are given in Problem 3 of Assignment VI. The project topics need not be confined to the ones listed below. The programs may be implemented in C, C++, perl, and shell scripts (or a combination). Java and other high-level languages are not allowed.*

Topic 1: Wireless Ad Hoc Forwarding

Suppose we have a number of wireless stations communicating with each other in ad hoc mode, where the aim is to send files amongst each other. For concreteness, assume there are three stations A, B, and C (one laptop and two pocket PCs). A and C may be too far apart for IEEE 802.11 to successfully transmit data, but if B is wedged in-between then A may use B as a relay station to reach C (and vice versa). The aim is to build a wireless file transfer client/server application (call it `wftp`) that uses sensing and tunneling to achieve this goal. One may assume that initially a file containing the IP addresses of all ad hoc wireless stations are given in a file at all stations (how it gets there is a bootstrap problem). A, to send a file to C, would execute

% `w_ftp` *IP-address file-name*

with C's IP address. The file transfer application at A would first check if C's address is valid (a member of the ad hoc group) and directly reachable from A. If so, it would directly transmit to the file transfer server process, `w_ftpd`, running on C. If not, it has to first relay through B, which means tunneling through B. That is, A's `w_ftp` sends to B's `w_ftpd` which, in turn, forwards to C's `w_ftpd`. Since A may have other potential relays besides B in the same ad hoc group, it may first send a query message to the potential relays one-by-one, receiving both reachability and performance related information from the candidate relays. The performance related information may be delay values from `ping` or signal strength values obtained from the WLAN driver (`iwconfig` is the command-line wireless extension to `ifconfig`). Benchmarking entails using 3–4 wireless stations that are placed in a line or diamond-like configuration so that relaying is necessary, then measuring the throughput achieved by the `w_ftp` application through clever choice of forwarding. Baseline benchmarks include throughput tests, as a function of distance, between two directly reachable stations. Then the overhead and slow-down stemming from forwarding when a relay station is wedged in-between. Since TCP ACK traffic—in the opposite direction—may collide with TCP data traffic at the relay station (an instance of the "hidden station" problem), other interesting benchmark variations are to compare throughput, under the same set-up, with/without RTS/CTS turned on which is a simple matter of choosing the RTS frame threshold. `w_ftp` should work correctly: files are transported with no corruption or missing pieces. The interesting question is: how well does it do its job. Laptops and pocket PCs may be borrowed for benchmarking from my lab. Most of the code development can be done in Xinu, except if signal strength is to be used directly for next hop selection. Porting C/C++ code to pocket PCs (they run Familiar Linux) requires cross-compilation at open sites operated by `handhelds.org` (supported by Compaq/HP), a straightforward matter.

Topic 2: WLAN Handoff Performance

Mobility support through WLAN handoff is achieved by mobile host/AP reassociation in conjunction with wireline Ethernet frame forwarding via bridging. A potentially interesting question is: what impact does handoff overhead between APs exert on UDP-based real-time streaming performance and TCP-based file transfer performance? For research purposes, we operate a 6-AP 802.11b WLAN testbed (SSID "Park") in the CS Building (2 APs per floor) that are connected through an Ethernet switch. Performance evaluation would entail running a UDP CBR sender/receiver in close proximity under a given AP, then move the sender (or receiver) around the 3 floors from AP to AP triggering reassociations. From the logged throughput measurements—including jitter in the delay time stamps—and hand-logged (approximate) position in the CS Building the magnitude of the impact of handoff on CBR traffic (representative of VoIP) may be discerned. The same goes for TCP-based file transfer where backoffs may be triggered during handoff which have performance repercussions after the functional handoff is completed. Subtleties such as, "does a noticeable/reliable performance degradation materialize prior to handoff as a mobile approaches a 'grey' zone?" pose interesting questions to understanding how bad or good the current handoff scheme is, and where

it may be lacking. Associations can be tracked at the APs using SNMP logs/probes that give a precise indication of handoff and resultant performance.

Topic 3: The Shape of Hot Spot Traffic

Using a powerful WLAN sniffer that records all relevant performance metrics—number of active wireless stations, their LAN throughput, per packet/frame signal strength, error rate, payload type, payload size, nested payloads requiring deep packet inspection, etc.—one may evaluate the static and dynamic make-up of WLAN Internet traffic at a popular hot spoterror rate, payload type, payload size, nested payloads requiring deep packet inspection, etc.—one may evaluate the static and dynamic make-up of WLAN Internet traffic at a popular hot spot such as the Village Coffee Shop. Of course, any non-protocol related user payload information must be discarded. From the log files, the aim is to do a careful analysis of the data that exceeds the sometimes coarse conclusions that such tools already carry out. In the static case, statistics on what fraction of traffic is UDP, TCP, and application layer protocols above UDP/TCP (such as HTTP), what is the distribution of frame/packet sizes, what is the lifetime distribution of TCP sessions, how many bytes do their transfer ("elephants and mice"), what is the throughput of the WLAN, what is the throughput share of individual wireless stations, and so forth are stats relevant for future engineering of WLAN Internet access networks. On the dynamic front, what's the pattern of TCP session arrivals over time, what's the pattern of UDP flow arrivals over time, how does total WLAN throughput change over time, what does individual WLAN share change over time, are some of the questions relevant to evaluation WLAN performance under Internet workloads. Experimentation would entail spending a couple of hours during a busy (and not-so-busy) period in a hot spot, logging the measurements, then performing off-line analysis. The same can be repeated in the CS Dept.

Topic 4: Greedy UDP Congestion Control

This would be comprised of two steps: first, build a negative ACK (NACK) oriented ARQ protocol on top of UDP that works well if packet losses are spurious. Then considering that it will share bandwidth with TCP flows, "greedify" it so that it will—by exploiting TCP's gentlemanly behavior—monopolize bandwidth giving only a trickle to TCP. Note that this can also be viewed as a form of denial-of-service attack. As in Problem 6 of Assignment VI, the subtlety lies in not "shooting oneself in the foot" by being overly greedy to the point where one's own throughput declines. The baseline demo is to run 6–7 large file transfer TCP sessions on separate machines that send to clients on the same host. This will cause packet losses at the kernel/NIC buffer of the receiving host and/or losses at the Ethernet switch's common output port. Using end-to-end measurement logs and `tcpdump`, total TCP throughput as well as individual throughput share would be ascertained. Then one greedy UDP file transfer session is introduced replacing one of the TCP sessions. Total system throughput as well as individual throughput share are evaluated. Then the fraction of greedy UDP flows is increased until all flows are greedy UDP flows. If the protocol is well-designed, then the all UDP-flow configuration should match the throughput of all-TCP connections, indicating that no self-congestion (a form of "tragedy of commons") is taking place. Then, as in Problem 4 of Assignment VI, it would be interesting to benchmark the UDP application on the commodity Internet (at least outside the Xinu lab confines) and compare its throughput and completion time against TCP-based file transfer (`ssh` or `ftp` due to signaling overhead should not be used; instead write a simple TCP-based file transfer app). Of course, the UDP-based ARQ must be shown to work correctly (all bits of a file, without corruption or holes, are tranferred without error).

If you are already familiar with kernel level programming, then your solution to Problem 6 of Assignment VI may be directly implemented. Alternatively, an elegant implementation strategy is to use `netfilter` support in Linux 2.4+ to install callback functions in IP hooks that mangle incoming/outgoing IP packets, manipulating the TCP fields transparent to TCP and the application above. In the latter, one would first have to check if the particular greedification scheme can be realized in this manner.

Topic 5: Pseudo Real-Time Audio Streaming with ARQ

Extend the pseudo real-time audio streaming application of Problem 5, Assignment VI, so that missing packets (identified by sequence numbers) can be filled in through NACK oriented ARQ before playback is due. For example, if the current buffer level is at $Q(t)$ bytes and a packet has been found missing, then $(Q(t)/\textit{packet-size}) \times \textit{pb-sp}$ (msec) is the time available to the receiver to recover the missing packet before it must be played back. Thus depending on the current RTT, this may or may not be meaningful to attempt. The initial playback delay *pb-del* and target buffer

level *target-buf* need to be set to counter-act network disturbances leading to packet drops in the envisioned operating environment. A baseline benchmark entails running the application over an application layer router—a simple forwarding app that uses tunneling to forward IP packets to their final destination—where packets are dropped with probability p. This may also be done at the receiver (as in previous assignments), but in this instance dropping at a separate router entity is preferred. Of course, by sending significant aggregate UDP cross traffic from other machines to the same receiver losses at the receiver buffer or switch may be triggered. Another interesting benchmark is to run the application over the commodity Internet and determine if it is able to effectively fill in holes in a timely basis and hide these holes from the user during continuous playback.

Topic 6: RSVP Guaranteed Service using Netfilter

RSVP (ReSerVation Protocol) is a receiver-oriented signaling protocol running on top of IP that allows resources to be reserved along an end-to-end path on behalf of a traffic flow. An RSVP reservation request is issued by a receiver containing a flow descriptor that is comprised of a flow and filter spec for identifying the TCP/UDP packet flow and the QoS that it should receive at participating routers. The packet flow is identified by a triple—destination IP address, port number, and protocol ID—and QoS specs include identifying service classes (e.g., WFQ) at routers. Although RSVP, by default, uses raw sockets (i.e., runs directly over IP) with protocol number 46, UDP encapsulation may be used at end systems to side-step raw sockets which typically requires root privilege. The aim of this project would be to design and implement a simple, bi-directional "bandwidth reserver" for VoIP (and video conferencing applications) that, upon invocation, sets up a bi-directional channel that is protected from cross traffic and congestion at routers. Its usage would be something like

% `my_reserve` *IP-address port-number protocol-ID bw application-name application-command-line-args*

where *protocol-ID* is 0 (UDP), 1 (TCP) and 2 (both), *bw* is requested bandwidth (Kbps), and *application-name* and *application-command-line-args* are the application to be run and its command-line arguments. In this setting, we do not concern ourselves with policies governing admission control—who is allowed to reserve how much—and just focus on setting up the appropriate QoS channel. Signaling may be implemented in raw IP or UDP encapsulation. Benchmarking involves running the bi-directional sender/receiver application (e.g., bi-directional CBR traffic generator) under `my_reserve` on two Linux PCs connected to a IP-over-SONET backbone comprised of 9 Cisco 7200 series routers. These routers may be bombarded with UDP cross traffic to cause heavy congestion. The aim is to show that the application traffic is shielded from such congestion, receiving its requested bandwidth.

Topic 7: Packet Filter for Worms

Worms are network viruses that travel from machine to machine under the auspices of network protocols that then trigger security vulnerabilities nascent in server code such as buffer overflow. The aim of this project is two-fold: (i) to catalog most of the well-known worms (e.g., Blaster, CodeRed2, Slammer) which includes obtaining "specimens" that are put in a "jar" and identifying their signature, and (ii) implementing an application layer firewall (i.e., packet filter) that applies the filter rules to weed out malware in incoming traffic. The main issue is maintaining low overhead and high speed. Benchmarking entails sending various worms in different mixtures at full throttle (close to 100 Mbps) to a target packet filter and ascertain whether it can filter out all the bad traffic (zero false negative) while letting through all the good traffic (zero false positive). The latter requires nimble packet processing and signature computation. By looking at traffic logs, false negative and false positive rates may be ascertained and the adequacy of the system design evaluated with respect to its efficiency.

Topic 8: Many Others

If none of the above strike your interest, then there are many others that can be discussed, including those that you may choose to specify and propose. There is no need to do something that is boring. The projects are intended to put your hereto acquired skills to the test in a more research oriented, less confined, creative setting.