Revisiting multimedia streaming in mobile ad hoc networks[*]

Peng Xue Surendar Chandra
University of Notre Dame, Notre Dame, IN 46556, USA
(pxue,surendar)@cse.nd.edu

ABSTRACT

Mobile ad hoc networks have been the subject of active research for a number of years. This paper investigates the feasibility of using such networks for transmitting multimedia streams. We observe that wireless network IO operations can be expensive (e.g., programmed IO cost, energy to operate wireless). Moreover, compared to nodes in infrastructure networks that either read or write network traffic, ad hoc traffic requires the intermediate node to perform many expensive network operations *twice* (read and then resend) and on behalf of other nodes. This observation raises an important question for the ad hoc community, should they a) demand that ad hoc routers support some minimum hardware resources (for example, full DMA support, twice the battery capacity)?, b) force an end-to-end resource management scheme that cooperatively reduces the network flow to half of what can be serviced by the weakest link? This would ensure that no intermediate node would see enough traffic to overwhelm them? or c) require that the local nodes protect themselves from transit traffic? This paper explores the last mechanism in order to provide some control over the resource consumed without a major revamp of existing operating systems or requiring special hardware. We implement our mechanism in the network driver and present encouraging preliminary results.

Keywords

MANET, multimedia streaming, resource management

1. MOTIVATION - OBSERVED SYSTEM BEHAVIOR

Mobile ad hoc networking (MANET) community provides us with a wealth of technologies that enable the source and the destination nodes to route the data through a number of intermediate forwarding nodes. There has been extensive research results in experiences with MANET [10], routing algorithms [19], capacity issues [13], energy aware routing and streaming mechanisms [8, 21, 2, 5] etc. Liu et. al. [14] are attempting to provide a single system image for ad hoc scenarios in MagnetOS.

Our primary goal in this work is to investigate the suitability of ad hoc networks for streaming multimedia contents. First we experiment with a wide variety of operating systems and hardware setups to show that there is a mis-match between expectations of the local system designers and the demands placed by ad hoc traffic. Decisions such as the choice of programmed IO and available battery capacity can make the system unable to transmit even modest multimedia streams. Building a system is an exercise in trading off device capabilities for cost and other constraints. Compromises in system resources (such as lack of DMA and battery power that are made for cost and weight reasons) are exacerbated by ad hoc networks which force these operations to be performed twice for each forwarding operation. There are three alternatives to solving this problem:

- *Decree that intermediate ad hoc routers should have better resources than what is required for end hosts*
- *Develop a global resource management policy that manages the resource consumption of the entire ad hoc network as a single schedulable entity*
- *Develop mechanisms that work with existing hardware and operating system mechanisms to provide better resource control*

These questions need to be addressed in order to make multimedia streaming a viable option in wireless ad hoc scenarios. In this paper, we choose mechanisms that can work with existing hardware and operating system limitations. In our system, the intermediate nodes manage their local resources by selectively dropping forwarding packets. The challenge is to choose the right set of packets. We show that kernel based policies for rejecting packets may not provide adequate resource control. We then describe our driver based approaches that can allow the intermediate router to control the transit flows without completely revamping the operating system. We present some encouraging preliminary results.

The rest of the paper is organized as follows: Section 2 shows our experimental quantification of the effects of ad hoc traffic

[*]This work was supported in part by the U.S. National Science Foundation (IIS-0515674 and CNS-0447671)

on the routing node as well as the effects on the end-to-end traffic. Section 3 discusses different mechanisms to attack the problems. Section 4 describes our approach. We place our work in context of related work in Section 5 and conclude in Section 6.

2. QUANTIFYING THE OVERHEAD FOR AD HOC TRAFFIC

2.1 CPU overhead for ad hoc traffic

We conducted experiments to quantify the effects of the forwarding traffic on a wide variety of hardware and software setups. Our setup is designed to represent a typical deployment using laptop class hardware. Our goal is not to stress the network; we experiment with network traffic that should be well within the capabilities of these class of machines. For our experiments, we investigated Microsoft media, Real and Quicktime multimedia formats. We streamed the 1:59 minutes long reference Wall (movie) theatrical trailer (also used in [4]) for our media stream. We used Pentium 4 Mobile (2 GHz) and Pentium M (1.6 GHz) laptops with 512 MB memory and 800 MHz Powerbook G4 (running Mac OSX 10.3.4) with 1 GB memory. The Powerbook routers ran Mac OSX 10.3.4 while the Pentium routers ran FreeBSD 4.9/5.x, Redhat 9.0 and Windows XP. We ran Windows XP for the browsers and Windows 2000 Server running Windows Media service, Realserver 8.01 and Apple Darwin Server 4.0 for the servers. Wireless access was provided by Apple airport for the Powerbook and Orinoco Silver, Dell TrueMobile 1150 mini-PCI and Linksys WPC11 v3 for the Pentium laptops on dedicated wireless channels. We use a 11 Mbps wireless network which supports throughput of around 5 Mbps. All the routers were running the native windowing system and other system processes (e.g. virus scanners). The systems were not running any other interactive applications; any of which would demand the use the CPU resources. We only show few representative results from these various experimental setups.

We tabulated the percentage CPU load placed on the intermediate routers for different streams, operating systems and device setups in Table 1. From Table 1, we note that the transit traffic consumes a significant amount of CPU resources across the various experimental setups. The load placed is not a function of spurious interaction between inexpensive wireless network interfaces (WNICs) and drivers; the load was high across vendor supplied drivers as well as open source drivers, operating systems, architectures (Mac and PC) and WNIC cards and interfaces (PCMCIA and mini-PCI). We note that high quality MS media streams can consume about 60% of the CPU cycles for interrupt processing. For the same stream bandwidth, formats such as Real and Quicktime, with more number of network packets can consume relatively higher CPU resources (e.g., a 256 kbps stream, MS media, Quicktime and Real transmit 2362, 3551 and 4383 packets respectively [4]).

We also note that local energy management policies that use dynamic voltage scaling and reduce the CPU clock frequency [3] further exacerbate this problem. Reducing the clock frequency for Windows XP has the effect of dropping network packets without much saving in processor load. On the other hand, FreeBSD saturates the CPU as the CPU clock is lowered (WinXP and FreeBSD at 1.2 GHz). Stream bandwidths of more than 500 kbps can consume more than 80% of the CPU resources. In fact, a 2000 Mbps stream (or two independent 500 kbps streams) can spend 100% CPU cycles for processing ad hoc traffic. Energy conservation mechanisms for the end nodes that shape the traffic to be transmitted in bursts [20, 5, 9] are expected to exacerbate this problem with high router CPU load spikes during these bursts.

2.1.1 Understanding the poor CPU load behavior of ad hoc traffic

In order to fully understand the effects of transit traffic, we profile the FreeBSD 4.9 operating system on the router. The laptop was running at 2 GHz. We measured the time it takes for a packet to be read from the wireless NIC card into the kernel and then resent back into the NIC card using the *microtime*() kernel function. We repeated the experiment for the Dell TrueMobile 1150 mini-PCI card and Cisco Aironet 350. We analyzed the system for a 1500 byte UDP packet. The UDP packet took $2830 \mu sec$ to transit from the network card back to the network card. The forwarding process consumed about $118 \mu sec$ for preparing to read the packet and $1314 \mu sec$ in programmed IO routines to read the data off the cards. It spent $32 \mu sec$ for route processing within the kernel before spending another $1386 \mu sec$ to write the packet to the network card.

On further investigation of the network driver source code and specifications of the network cards used in our study, we note that these cards require the kernel to use programmed IO to read and write data from the wireless card. Essentially, the bottleneck is the CPU used for programmed IO. The limitations of programmed IO are well studied and understood [18]. Previous work [15] has addressed some of the limitations of programmed IO in the context of servers. However, in an ad hoc scenario, the increased load is created by transit traffic. *Unlike servers, the system can choose to not spend resources on ad hoc traffic and drop the transit traffic.* Note that programmed IO operations are charged as a kernel resource and are not preemptible.

2.2 Energy overhead for ad hoc traffic

Also, in [5], we investigated the energy overhead for forwarding ad hoc traffic. We showed that the routers consume more energy than the browser or the server nodes. The power consumption of the router nodes are higher as the packets are processed twice. The routers receive traffic and resend them; while the servers and browsers only send and receive data, respectively.

2.3 End to end network behavior for ad hoc routing

The end to end behavior of the ad hoc traffic also suffer in our setup. We plot the normalized time delay between the packet leaving the server and the time it was received at the browser (using tcpdump traces at the server and browsers) as a cumulative distribution in Figure 1. We normalize with respect to the delay experienced by the first packet in the stream; we are concerned with the variance rather than the actual delay through the network. The router was running

System setup	MS media				Real			Quicktime	
	2000 kbps	768 kbps	256 kbps	128 kbps	512 kbps	256 2kbps	128 kbps	256 kbps	128 kbps
MacOSX	39%	20%	13%	9%	57%	31%	11%	17%	13%
WinXP (2GHz)	62%	21%	13%	13%	83%	80%	28%	28%	23%
WinXP (1.2GHz)	69%	21%	8%	5%	55%	35%	20%	27%	14%
Win XP ($WNIC_2$, 1.2 GHz)	64%	20%	11%	6%	35%	35%	18%	21%	10%
FreeBSD 4.9 (2GHz)	59%	20%	6%	4%	21%	20%	11%	22%	11%
FreeBSD 4.9 (1.2GHz)	100%	77%	24%	10%	83%	73%	43%	65%	24%

Table 1: CPU load on intermediate routers for forwarding multimedia streams

FreeBSD 4.9 at 1.2 GHz speed. We repeat experiments under three scenarios: text console with no graphical front end, idle X front end and X front end with continuous MP3 playback using XMMS reach with no normalized delays. Figure 1(a) plots the normalized delay for a 56 kbps Microsoft media traffic. We note that the routing adds a variance of about 10 msec; easily buffered and adapted by the multimedia browser. Figure 1(b) plots the normalized delay for a single 768 kbps MS media traffic while Figure 1(c) plots the normalized delay for two 768 kbps MS media traffic traversing the router nodes to two different clients. We note that as the stream bandwidth increases, there is more variance in the delays. The effect is further pronounced by the router running GUI applications for their local consumption. The routing completely breaks down for routers handling two high quality streams (Figure 1(c)). Since the streams are variable bit rate in nature [5], the load on the router changes and the multimedia system is not able to effectively adapt to the streams. Also, the router node becomes completely incapacitated during peak activity. Note that we use a 11 Mbps wireless network which supports throughput of around 5 Mbps; network capacity itself is not the bottleneck.

To summarize, we analyzed the effects of multimedia streams on the ad hoc routers. We noted that multimedia traffic can place tremendous demands on the intermediate routers. Even for moderate streams ($<$ 500 kbps), the offered load can incapacitate the routers. The routing traffic interferes with local energy resource management policies. The CPU load on the intermediate node also increases the variance of the streams, changes which are too quick for commodity streams to adapt and lower their quality requirements. Essentially, the stream was not acceptable for the end nodes or the intermediate router because it forces the intermediate nodes to perform operations which are known to be expensive (e.g. programmed IO, wireless operation), twice. These expensive operations are also performed on behalf of other nodes.

3. NATURE OF THE SOLUTION

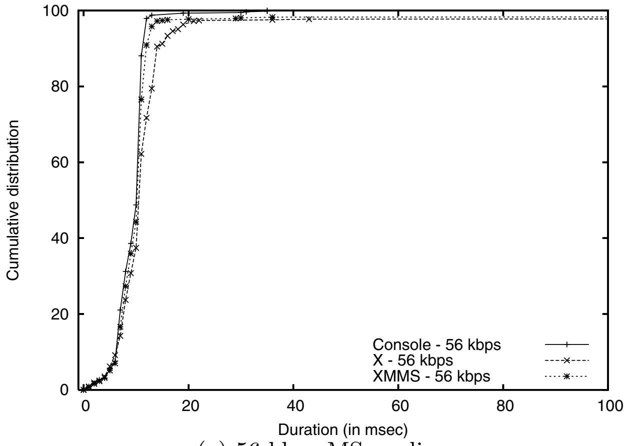
Ad hoc networking technologies essentially benefit other nodes with the hope that the other nodes will eventually help the current node for its future network demands. The previous section paints a grim picture; it suggests that ad hoc network scenarios might not be appropriate for multimedia streaming. Building a system is an exercise in trading off device capabilities for cost and other constraints. These compromises are exacerbated by ad hoc networks which force expensive operations to be performed twice for each forwarding operation. There are three alternatives to solving this problem:

- *Decree that intermediate ad hoc routers should have good resources:* Perhaps, the solution to the woes pointed out in the previous section is to demand that the ad hoc routers be resource rich. We can demand that the ad hoc routers should have higher battery capacity in order to read and resend traffic. Such demands add to the weight and cost of the ad hoc routers.

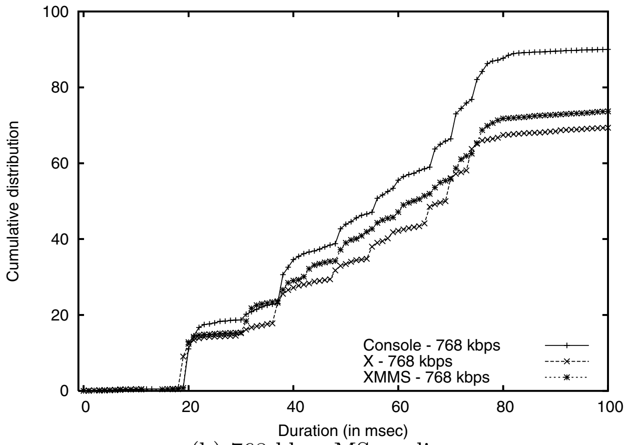
  Previous work on servers had noted the fundamental limitations of programmed IO and recommended DMA as the solution. DMA offloads IO processing to a separate DMA controller, freeing the main processor for other tasks. One possible solution for our ad hoc scenario is to demand that ad hoc routers use DMA based IO. For mobile devices, we argue against this solution. As noted by [7], programmed IO has better cache behavior for packets that will be immediately forwarded. Also, DMA controllers essentially require more hardware resources to solve the IO problem. Using DMA also requires further complexity of the OS; DMA scatter-gather operation requires more interrupt processing. For a mobile device, we argue that a better place to invest in hardware resources is in the CPU where the additional resources can be used by local processing rather than in dedicated I/O processing. Moores law gains in hardware do not necessarily invalidate this argument.

  Besides, these mobile devices were functioning acceptably for the local non ad hoc user. Ad hoc traffic should not require that the intermediate router in fact provide server like resources. The promise of ad hoc networking is to use mobile devices in an ad hoc fashion, not to require certain capabilities which are typically not found in commodity mobile devices (this observation was based on our experiments with a wide range of hardware).

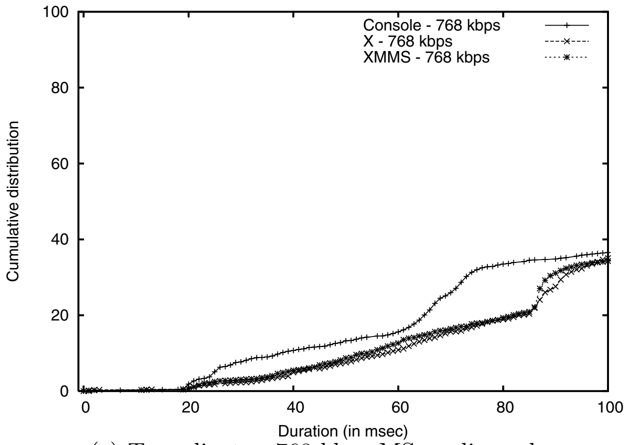
- *Develop a global resource management policy that manages the resource consumption of the entire ad hoc network as a single schedulable entity:* Another tempting alternative is to address this problem as a global resource management issue. We can unify the resource accounting and management across the entire ad hoc deployment. As the load on the intermediate router increases, it can explicitly control the source data rate in order to manage the resources on all the participating nodes. MagnetOS takes this design philosophy. This approach changes the flavor of ad hoc networks from a collection of nodes cooperating to provide useful services to a combined entity that services a particular application.



(a) 56 kbps MS media



(b) 768 kbps MS media



(c) Two clients - 768 kbps MS media each

Figure 1: Cumulative distribution of normalized delays introduced by the router running FreeBSD 4.9. The routers were running with no windowing system (Console), with X windows as well as X windows + xmms playing a locally available MP3 song

- *Develop mechanisms that work with existing hardware and operating system mechanisms in order to provide better relief:* In this research, we take a more modest approach and analyze how we can make OS software work better on existing hardware for ad hoc scenarios.

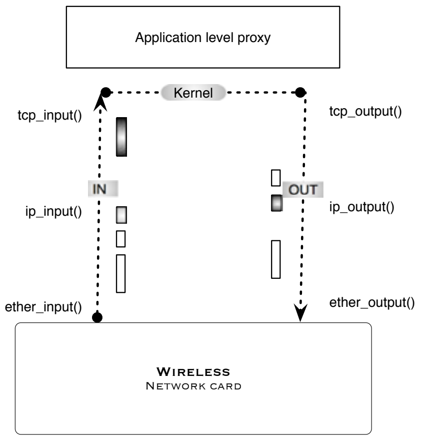
4. LOCAL RESOURCE MANAGEMENT FOR EXISTING HARDWARE

This paper focuses on resource management policies that are appropriate for the existing hardware. In general, the resource management components for forwarding network packets is illustrated in Figure 2(a). The intermediate router spends resources to read the packet off of the network card, performs some in-kernel processing and then sends the packet back to the wireless network card. Application level proxies might also read the packets from the kernel to perform further processing. The key to controlling the resources consumed by the forwarding process is to measure the amount of resources consumed in each phase and develop software strategies for saving those resources.

4.1 Local CPU resource management

We analyze the software changes required in order to better control the CPU resources consumed by ad hoc traffic. Earlier, we measured the time to forward a packet through the kernel at $2830\mu sec$ for a 1500 byte packet. We explore mechanisms to reduce this overhead. The bulk of the overhead was in the programmed IO routines with about $32\mu sec$ spent in kernel forwarding. For these hardware, the only viable option for resource control is to develop techniques to reduce the PIO overhead by dropping packets. Dropping packets would eventually force the source to reduce its transmission rate. The challenge is to drop packets that are the least disruptive. For example, dropping a frame that is part of a fragmented datagram would force the entire packet to be dropped, even though the system already paid the cost to transmit the other fragments. The end user can influence any policy by dropping packets using kernel firewalls. Dropping a packet using such firewalls would still take at least $1444\mu sec$. We desire a mechanism that can let the network driver itself drop this packet. The network driver should drop the packet after performing the least amount of programmed IO.

First we analyze the performance of policies that drop packets in the kernel (using firewalls) as well in the network driver. The system is configured using kernel firewall (*ipfw*) to reject any transit traffic but allow any legitimate local traffic. To achieve the same effect inside the network driver, we modified the driver to only read the first 34 bytes (14 and 20 bytes of Ethernet and IP headers, respectively) and reject any transit traffic: the next read operation flushes the remaining packet data in the card. Local traffic is forwarded to the kernel as usual. For our experiments, we use three Pentium laptops running at 2 GHz and measured the CPU load using *top*. We modified FreeBSD 4.9 kernel for our experiments (FastForward option was enabled). The server generated UDP streams of fixed size at a constant rate using *iptraffic* for our experiments. We plot the percentage CPU load for various rates of network traffic and packet sizes in Figure 3. From Figure 3, we note that the CPU overhead for dropping packets in the kernel depends on the packet arrival



(a) Resource management for forwarding traffic

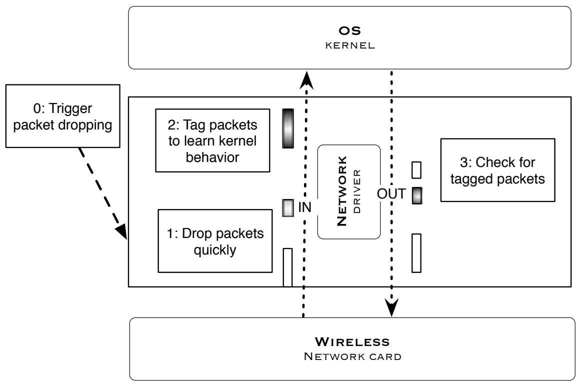

OS
Kernel
0: Trigger packet dropping
2: Tag packets to learn kernel behavior
Network driver
IN
OUT
3: Check for tagged packets
1: Drop packets quickly
Wireless
Network card
(b) CPU resource management

Figure 2: Resource management for forwarding network traffic

rate and size. Dropping the packets at the driver can allow for better resource management. The overhead of dropping packets in the driver is not dependent on the packet size. Dropping packets in the network driver can save significant amounts of CPU load for all the packet sizes.

4.2 Design of our modified network driver

In the previous section, we showed that the network driver can be a good place to drop network packets in order to manage the overall CPU load induced by forwarding traffic. A wide variety of network packets are destined for transit nodes, including local unicast, broadcast and multicast messages. In this work, we are only interested in the behavior of transit traffic. The challenge is to drop packets that can quickly force the end hosts to adapt to a lower resource requirement level. Ideally, the driver would require multimedia stream semantics (e.g., Quicktime, Microsoft media) in order to choose to drop the right set of packets. However, the network driver does not possess all the network protocol stack information. Moving this functionality into the driver will not only violate the network layering principles, but will also slow down network processing for other local traffic. So, we are designing a network driver that *learns* the firewall rules (and kernel behavior) by watching the behavior of transit packets. Our driver does not depend on whether the user implements mechanisms to drop packets inside kernel firewalls or in application level proxies. Typically, kernel firewalls are a good place to perform packet filtration with policy decisions themselves made in application level proxies. Our solution is illustrated in Figure 2(b)).

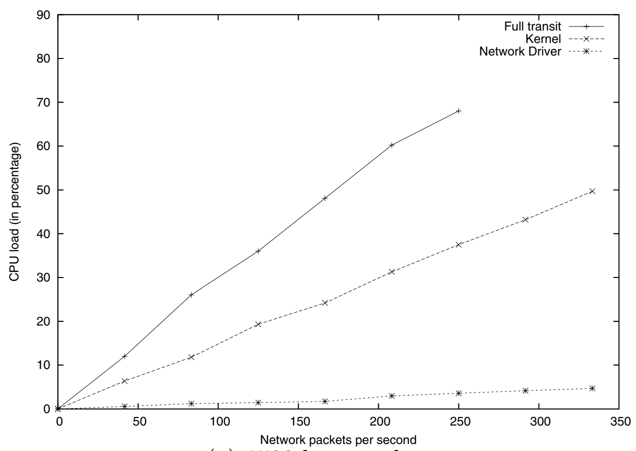
4.2.1 Step 1: Drop packets early

Whenever a packet arrives at the network driver (*wi_rexof()*), we analyze the packet to identify whether this packet belongs to a transit flow which the kernel is likely to drop. The information about what packet to drop is collected in Step 3. Note that the driver will not implement complex policies that is aware of the streaming mechanisms to drop the appropriate packet. Such policies will be implemented at the kernel or application level. The driver packet rules are far simpler than kernel firewalls; operating on packet byte offsets.

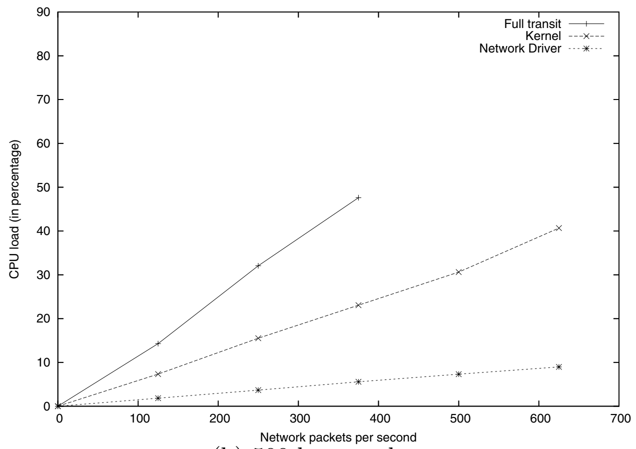
Programmed IO costs are directly proportional to the amounts of bytes read from the WNIC card. Hence, we make the decision by reading as few bytes of the packet as possible. If the packet was not likely to be dropped, then the entire packet is read and forwarded on to Step 2 (programmed IO duration does not depend on CPU frequency. Hence we can potentially reduce the CPU clock frequency to save energy). Note that in general, multimedia network packets are best-effort traffic. False positives lead to end-to-end service disruption while false negatives reduce the ability of local node to manage its resources. Hence, false-positives where packets are wrongly identified (and dropped) are more tolerable than false-negatives where packets that should have been dropped are transmitted to the kernel (consuming unnecessary CPU resources).

4.2.2 Step 2 and 3: Learn what packets to drop

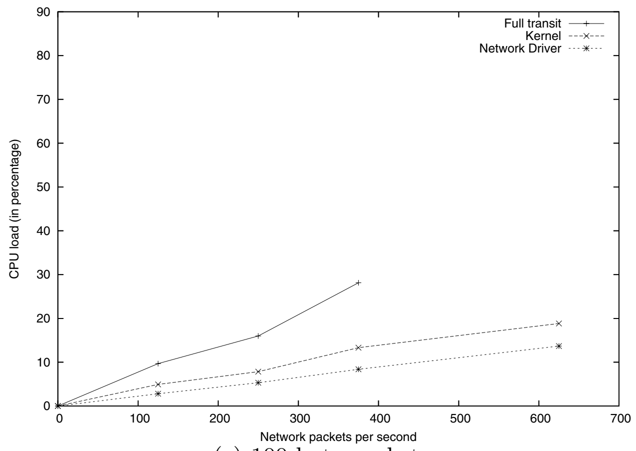
The network driver should learn the behavior of kernel firewall. Note that the driver itself does not understand the packet format. For new transit flows, the driver notes some identifying markers of the packets and starts a timer to see if this packet reappears on the *OUT* side (in *wi_start()*). If the marked packet reappears, then the driver learns that this particular flow is being forwarded and so continues to forward packets for this flow in the future. If the packet was not resent by the kernel within a certain duration, then the driver learns that this flow is potentially being dropped by a kernel firewall (being best-effort traffic, the packet might have been dropped without explicit firewalls). After n such drops, the driver will itself drop subsequent packets for this flow (in Step 1). The choice of n is a tradeoff between the false negative rates and false positive rates? Good values for n for a particular hardware setup will be experimentally measured. Also, the duration between Steps 2 and 3 depend on a number of factors that are unknown to the driver. It depends on the system load as well as any CPU clock frequency changes for potential energy savings. The kernel driver also does not have access to interruptible timers. Keeping track of a large number of packets adds to the overhead for packet matching in Step 3. Preliminary results indicate that keeping track of one or two packets was enough for practical media streaming for a single transit traffic. Also, the ker-



(a) 1500 byte packets



(b) 500 byte packets



(c) 100 byte packets

Figure 3: CPU load for varying packet size and forwarding rate

nel firewall rule changes are not notified to the driver; the driver will periodically forward (duration will be experimentally measured) a matching flow to probe the kernel firewall. This period is chosen as a compromise between unnecessary (and resource intensive) probes and a latency between user-initiated change in firewall rules and any noticeable effects of the new rules.

The driver requires mechanisms to identify whether a packet belongs to a particular transit flow in order to drop packets quickly. Our mechanism should not depend on the driver understanding the semantics of IP datagrams. For our work, we are investigating IP identification (IDENT) field [17] as the potential differentiator. IDENT is a two byte field starting from the fifth byte of an IP datagram. This field was designed for fragmented packets. Our preliminary analysis of a large number of flows showed us that in practice, the IDENT field is a monotonically increasing number for a given source node. Based on this property, we can differentiate different flows from a single host within a small target window. Potentially, we only need to read 20 bytes (14 for Ethernet header and 6 to reach the IDENT field) of a frame in order to identify a network flow.

Presently we are building our driver that uses the IDENT field to tag potential packets as well as identify flows to be dropped. We will report our experiences with this driver in time for the conference.

4.2.3 Least interference for the rest of the packets:

Our driver should cause the least interference to legitimate traffic. This might require that the drive be explicitly triggered by the kernel (Step 0) or that the driver always looks out to drop certain packets based on its understanding of the kernel resource management needs. The contribution of false-positives and false-negatives will be measured and their contribution reduced.

5. RELATED WORK

Managing router resources have been well studied in the context of software routers [16, 15, 11, 22] and active networks [12]. The Click modular router [11] assembled a software IP router using programmable *elements* for various configurable network processing. nanoProtean [6] investigated active networking mechanisms in a gigabit router by reducing the OS system overhead for processing packets. Scout [16] introduces the notion of paths for resource management. Banga et al. [1] proposed the generalized resource container mechanism to account for resource consumption outside the traditional process boundary. The CROSS software router work [22] investigated the OS isolation and packet classification mechanisms necessary for a programmable software router. Ad hoc scenarios differ from these systems in that the primary function of these routers are to perform interactive user tasks; the forwarding operation is basically an service to other peers. Hence, rejecting traffic earlier on is preferable to ensuring that packets are forwarded.

6. DISCUSSION

This paper argues that wireless ad hoc routers need to effectively manage the transit traffic; account for the resource consumption as well as reject unsustainable flows quickly.

Without such mechanisms, the transit traffic can place inordinate resource demands, sometimes incapacitating the nodes completely. We develop a driver-based mechanism that enables the driver to learn the behavior of kernel firewalls and application proxies, and ultimately to save resources by dropping unsustainable packets. Such mechanisms can also potentially benefit the system defend against denial of service attacks.

7. REFERENCES

[1] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 45–58, New Orleans, LA, Feb. 1999. USENIX Association, ACM SIGOPS.

[2] A. Bestavros and S. Jin. Osmosis: Scalable delivery of real-time streaming media in ad-hoc overlay networks. In *Proceedings of Workshop on Data Distribution in Real-Time Systems*, pages 214–219, Providence, RI, May 2003.

[3] T. D. Burd and R. W. Brodersen. Design issues for dynamic voltage scaling. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, July 2000.

[4] S. Chandra. Wireless network interface energy consumption implications of popular streaming formats. In M. Kienzle and P. Shenoy, editors, *Multimedia Computing and Networking (MMCN'02)*, volume 4673, pages 85–99, San Jose, CA, Jan. 2002. SPIE - The International Society of Optical Engineering.

[5] S. Chandra. Energy conservation in ad hoc multimedia networks using traffic-shaping mechanisms. In N. Venkatasubramanian, editor, *Multimedia Computing and Networking 2004 (MMCN '04)*, volume 5305 of *Proceedings of SPIE-IS&T Electronic Imaging*, pages 40–54, San Jose, CA, Jan. 2004.

[6] D. Craig, H. Kim, R. Sivakumar, V. Bharghavan, and C. Polychronopoulos. nanoprotean: scalable system software for a gigabit active router. In *Proceedings of IEEE Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001)*, volume 1, pages 51–59, Anchorage, AK USA, Apr. 2001.

[7] P. Druschel, M. B. Abbott, M. A. Pagals, and L. L. Peterson. Network subsystems design. *IEEE Network*, 7(4):8–17, 1993.

[8] L. M. Feeney. A QoS aware power save protocol for wireless ad hoc networks. In *Med-Hoc-Net 2002*, Sargegna, Italy, Sept. 2002.

[9] M. Gundlach, S. Doster, H. Yan, D. K. Lowenthal, S. A. Watterson, and S. Chandra. Dynamic, power-aware scheduling for mobile clients using a transparent proxy. In *International Conference on Parallel Processing (ICPP '04)*, pages 557–565, Montreal, Canada, Aug. 2004.

[10] E. Huang, W. Hu, J. Crowcroft, and I. Wassell. Towards commercial mobile ad hoc network applications: a radio dispatch system. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 355–365, New York, NY, USA, 2005. ACM Press.

[11] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[12] U. Legedza, D. J. Wetherall, and J. Guttag. Improving the performance of distributed applications using active networks. In *IEEE Infocom*, Mar. 1998.

[13] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee, and R. Morris. Capacity of ad hoc wireless networks. In *Proceedings of the 7th ACM International Conference on Mobile Computing and Networking (MobiCom '01)*, pages 61–69, Rome, Italy, July 2001.

[14] H. Liu, T. Roeder, K. Walsh, R. Barr, and E. G. Sirer. Design and implementation of a single system image operating system for ad hoc networks. In *International Conference on Mobile Systems, Applications, and Services (Mobisys05)*, pages 149–162, Seattle, WA, June 2005.

[15] J. C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *ACM Trans. Comput. Syst.*, 15(3):217–252, 1997.

[16] D. Mosberger and L. L. Peterson. Making paths explicit in the scout operating system. In *Proceedings of the second USENIX symposium on Operating systems design and implementation*, pages 153–167. ACM Press, 1996.

[17] J. Postel. *IP Protocol Specification*. DARPA and ISI-USC, RFC 791 edition, Sept. 1981.

[18] K. K. Ramakrishnan. Performance considerations in designing network interfaces. *IEEE Journal on Selected Areas in Communications*, 11(2):203–219, 1993.

[19] E. Royer and C.-K. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE PERSONAL COMMUNICATIONS*, pages 46–55, Apr. 1999.

[20] P. Shenoy and P. Radkov. Proxy-assisted power-friendly streaming to mobile devices. In *Proceedings of the 2003 Multimedia Computing and Networking (MMCN)*, Santa Clara, CA, Jan. 2003.

[21] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh. Power-saving protocols for ieee 802.11-based multi-hop ad hoc networks. In *Proceedings of IEEE INFOCOM 2002*, New York, June 2002.

[22] D. K. Y. Yau and X. Chen. Resource management in software-programmable router operating systems. *IEEE Journal on Selected Areas in Communications*, 19(3):488–500, Mar. 2001.