Exploiting Packet Header Redundancy for Zero Cost Dissemination of Dynamic Resource Information

Peter A. Dinda
pdinda@cs.northwestern.edu

Department of Computer Science
Northwestern University

Abstract. Packet headers exhibit considerable coding redundancy from both a theoretical and a practical standpoint. We propose to exploit this redundancy to create an additional communication channel between hosts as a by-product of normal packet transfers. This channel would be zero cost: the number and size of packets transfered would not change. Dissemination of dynamic resource information is a natural user of such a channel. In the paper, we begin by illustrating the significant theoretical coding redundancy of typical TCP/IP traffic using widely available packet trace data. Next, we discuss a number of ways to practically exploit this redundancy using only end-system changes and evaluate their prospects. Then we describe proof-of-concept experiments that we have conducted using a user-level network stack. Finally, we describe how common forms of resource information could be communicated using this mechanism.

1 Introduction

An important challenge in resource monitoring and prediction systems such as Remos [9], NWS [14], RPS [7], and GMA [13], is the dissemination of the data they collect. Sending the data from producers to consumers can potentially consume a significant portion of the available network bandwidth, especially when there are many resources being monitored. The general approach to this problem is to encode the data in the most efficient way possible given the constraints of the consumers [12, 15]. However, even if such methods are used and the bandwidth consumed is limited, it is still the case that additional packets are introduced into the network, packets that may interact badly with the non-monitoring traffic. Here, we propose an orthogonal technique that could be used to avoid introducing any new network traffic at all: exploiting packet header redundancy to transfer resource information.

The basic idea, summarized in Figure 1, is to piggyback additional information on the transport, network, and data link headers and trailers of regular packets by encoding the information into redundant fields. For example, in over 90% of IP packets, it

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, and EIA-0130869. The NLANR PMA traces are provided to the community by the National Laboratory for Applied Network Research under NSF Cooperative Agreement ANI-9807479. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

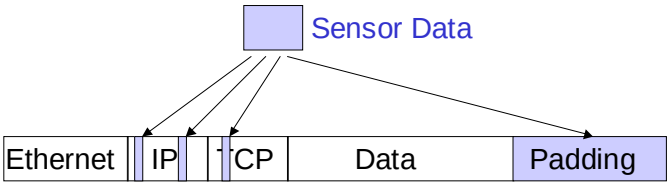

Fig. 1. Summary of approach.

is possible to trivially encode an additional 17 bits of information per packet. This is a sufficient number of bits to encode a CPU load sample, or a network bandwidth measurement. If the packet is also part of a TCP connection, as is extremely likely, it is also possible to recover the timestamp of the sample. Every keystroke in an ssh session could thus include a free exchange of timestamped load information. Every packet sent on a multicast audio stream could communicate resource information to all subscribers with no additional overhead.

Our purpose here is to demonstrate that this opportunity exists and can be exploited—to highlight this interesting communication channel. We focus on IP header redundancy, exploring both the theoretical limits of that redundancy and how it manifests in practical ways. Our conclusions are based on an analysis of a set of NLANR packet traces [10] collected from several PoPs. We then discuss a number of practical mechanisms to transfer data by exploiting TCP and IP header redundancy, and Ethernet padding. Finally, we evaluate these methods by implementing them in a user-level network stack and verifying that our information can indeed pass unmolested through various network configurations.

Transmitting information that happens to fit into the redundant bits of a single packet is the simplest way of using this communication channel. The channel has properties that are similar to a multicast session, and we believe that the forward error correction techniques that are sensible in multicast, such as Tornado codes [4], could be applied here. Our ultimate goal is to be able to diffuse resource information throughout the network using this and other techniques.

2 Header redundancy

We begin by discussing the amount of redundancy that theoretically exists within packet headers, concentrating on IP headers. Figure 2 is derived from an analysis of 28 packet traces, captured at 4 different PoPs within the NLANR PMA network in at different times during Wednesday, September 26, 2001. Each trace is roughly 90 seconds long and contains anywhere from 68,000 to almost 3 million packets. A PoP (Point of Presence) trace is interesting because it aggregates traffic from many different sources on two networks. It is not, however, representative of traffic within one network or LAN. We have repeated our analysis using a larger set of NLANR traces collected during Wednesday, June 5, 2002. We present results from the earlier traces and point out when the results for the later traces are different.

Description	Trace Name	Information bits/symbol	% redundant	Mutual Information (1 step) bits/symbol	%redundant
University of Buffalo OC3 to NYSERnet (Internet2)	BUF-1001462650	4.50	43.78	1.24	84.56
	BUF-1001473743	4.48	44.05	1.17	85.32
	BUF-1001486582	4.38	45.27	1.23	84.68
	BUF-1001497670	4.30	46.24	1.20	84.96
	BUF-1001506918	4.33	45.84	1.24	84.55
	BUF-1001519802	4.41	44.87	1.20	85.06
	BUF-1001529100	4.37	45.36	1.20	85.05
Columbia OC3 to NYSERnet (Internet2)	BWY-1001473743	4.64	42.00	1.25	84.38
	BWY-1001486581	4.56	43.04	1.23	84.57
	BWY-1001497671	4.53	43.35	1.28	84.06
	BWY-1001506918	4.46	44.22	1.18	85.29
	BWY-1001519803	4.53	43.43	1.20	84.94
	BWY-1001529100	4.64	42.06	1.19	85.10
	BWY-1001538343	4.65	41.83	1.28	83.97
Colorado State OC3 to Front Range GigaPoP	COS-1001473743	4.82	39.77	1.20	85.03
	COS-1001486582	4.71	41.23	1.26	84.20
	COS-1001497671	4.63	42.08	1.20	85.04
	COS-1001506917	4.62	42.17	1.20	85.02
	COS-1001519803	4.81	39.90	1.17	85.42
	COS-1001529101	4.86	39.24	1.19	85.17
	COS-1001538342	4.85	39.31	1.19	85.18
University of Memphis OC3 to Abilene	MEM-1001473742	4.29	46.34	1.09	86.32
	MEM-1001486581	4.23	47.18	1.20	85.01
	MEM-1001497670	4.19	47.61	1.24	84.46
	MEM-1001506918	4.19	47.62	1.14	85.20
	MEM-1001519802	4.39	45.17	1.23	84.58
	MEM-1001529100	4.20	47.52	1.24	84.56
	MEM-1001538342	4.34	45.81	1.25	84.40

Fig. 2. Redundancy in IP Traffic: Information content of IP header traces. Each trace is about 90 seconds long, and was collected on September 26, 2001.

Shannon entropy of IP header sequences

If we consider a packet header trace as being a stream of symbols, we can compute the stream's Shannon Entropy [11], the number of bits needed, on average, to represent a symbol given no knowledge of its relation to other symbols in the stream. We applied this measure of information content to the traces by creating a stream of consecutive IP headers from them and then interpreting that stream as a stream of bytes. With 256 symbols, we would need 8 bits per symbol if there was no redundancy. The third column of Figure 2 shows the entropy of each of our traces, and the fourth column shows the redundancy that this implies. On average, we need only about 4.8 bits per byte, and thus a 20 byte IP header is about 40% redundant. This suggests that on average we could conceivably encode 8 extra bytes of resource information into each IP header.

Field	Bits
TCP Headers	
Use reserved bits	6
Use TCP Acknowledgment field when ACK=0	32
Use TCP URG pointer when URG=0	16
Use TCP NOP padding	varies
IP Headers	
Use reserved TOS bits	2
Use reserved IP flag	1
Use ID field when DF=1	16
Use fragment offset when DF=1	13
Use IP NOP padding	varies
Data link layer	
Use Ethernet minimum packet size redundancy	varies
Total identified bits:	≥ 86

Fig. 3. Practical ways to exploit TCP/IP/Ethernet redundancy.

Mutual information of IP header sequences

The figure also shows that IP headers, when treated as a byte stream, show tremendous mutual information. Mutual information [3, Chapter 3] is an extension of Shannon entropy to streams where symbols do not occur independently. In the case of our traces, exploiting dependence from one packet to the next would let us encode the next byte in about 1.2 bits per byte, for a redundancy of over 85%! We could conceivably appropropriate 17 bytes out of every IP header to transmit resource information. While exciting, it's important to note that because of the aggregation that is happening here this is not really an estimate of redundancy we would expect from a packet header stream from an individual host. However, it is important to note that exploiting the redundancy from the TCP/IP headers in one packet to the TCP/IP headers in the next packet is extensively and successfully used when running TCP/IP over low-speed serial links [8].

The conclusion here is simple: there is a tremendous amount of redundancy in IP headers that could potentially be exploited to transmit resource information. We have not tested this yet, but we believe that this redundancy also extends to headers at other layers of the network stack.

3 Exploiting header redundancy while maintaining compatibility

Exploiting to the hilt the redundancy we have just illustrated would require a new encoding of the headers, which would immediately make a stack incompatible with other TCP/IP stacks. Evidence of practical redundancy is needed. We can not rearrange or recode the fields of a header. Instead, we need cases where it is acceptable to overload the existing header fields—to use their bits when the field is otherwise unused.

Figure 3 lists a number of approaches we have identified through which this might be possible. While we have identified over 86 bits per packet that could be exploited, we should point out that we have only studied the approaches at the IP level in any kind

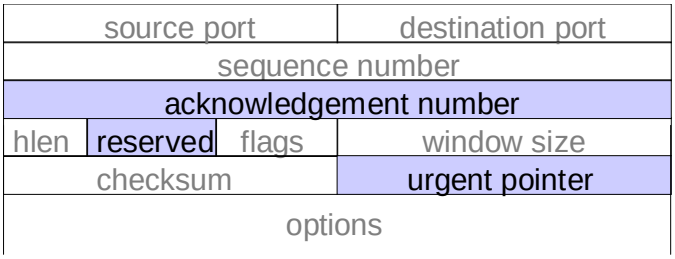

Fig. 4. Potential redundant fields in TCP headers.

of depth. There potentially in excess of 32 bits available at that level, but, as we show in the next section, we have only been able to successfully make use of 17 of them.

Transport layer (TCP)

The potential sources of redundancy in TCP headers are illustrated in Figure 4. A TCP header is 160 bits long, and may be followed by up to 320 bits of options [2]. The length of the header and option must be an integral multiple of 32 bits, additional NOP options being added until this is the case. One possible mechanism for for introducing additional information into the packet is to overload NOP. In effect, a NOP consists merely of an 8 bit type identifier and there are far fewer than 256 different TCP option types. Notice that overloading NOP would only help us send bits on those packets which use NOP options.

A TCP header contains a 16 bit pointer that points to urgent data within the packet. However, this pointer is ignored if the URG flag is not set. If an outgoing packet has URG=0, we could embed our own information into the pointer field.

Whenever possible, a sending TCP piggybacks an acknowledgment for data flow in the opposite direction when it emits a data segment. Every TCP header contains both a sequence number for the outgoing segment and an acknowledge number for some segment that arrived in the past. The 32 bit acknowledgment number is only valid when the ACK flag is set. We could embed 32 bits of our own data on each outgoing segment where ACK=0.

A final TCP mechanism we identified is to use the flags that are currently reserved. This would provide an additional 6 bits.

Network layer (IP)

The potential sources of redundancy in TCP headers are illustrated in Figure 4. An IP header is 160 bits long, with an additional 320 bits of options possible [1]. As with TCP, the total length must be an integral multiple of 32 bits and NOP options are used to pad to such a length. Here too we could consider overloading the meaning of NOP to add information to packets which have NOP IP options.

An IP packet can be fragmented into multiple IP packets in the network, with the receiving host having the responsibility of reassembling the original packet from the

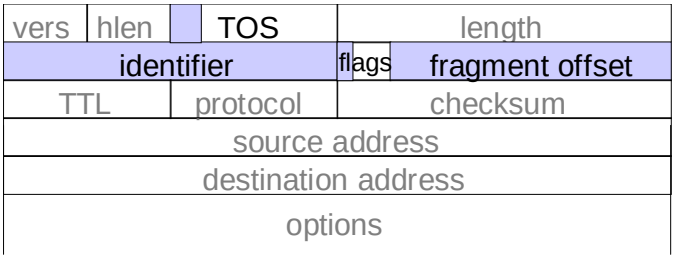

Fig. 5. Potential redundant fields in IP headers.

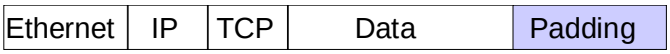

Fig. 6. Potential redundancy in Ethernet padding.

fragments. To support this, IP headers have a 13 bit field that contains the offset of the fragment with respect to the beginning of the packet, a 16 bit field that identifies the original packet, and a flag that indicates that a fragment is the last fragment. However, a sender can also make a packet unfragmentable by setting a don't fragment flag in the header. For outgoing packets which have DF=1, we could borrow the unused fragment offset and packet ID fields to transfer $13 + 16 = 29$ bits.

A last approach to use with IP is to borrow its reserved flags for information transfer. There are 3 bits to be gained here, one from the generic reserved flag and two from a portion of the type of service (TOS) field that is not currently used.

Data link layer (Ethernet)

At the data link layer, the most common hardware by far is Ethernet. Almost all LANs use Ethernet of some form. To traverse an Ethernet, an IP packet is embedded into an Ethernet packet. Because of Ethernet's media access control algorithm, the data portion of an Ethernet packet must be at least 368 bits long, but an IP packet can be as short as 160 bits. The difference is made up by padding the IP packet with zeros that will be removed by the receiving Ethernet device driver before the packet is passed up to IP. We could easily pad short packets with our information instead, as illustrated in Figure 6. Notice that the bulk of IP packets sent in an ssh or telnet session typically involve single keystrokes. The total length of such IP packets can potentially be as short as 328 bits (160 bit IP header, 160 bit TCP header, 8 bit keystroke), leaving room to insert extra information.

Prospects for these approaches

We examined our packet traces looking at the prospects our IP layer mechanisms and for some of our TCP layer mechanisms. Figure 7 shows the some of the results. Not surprisingly, the vast majority of the packets are TCP packets. We note that none of the

Trace	Packets	% of IP Headers that have						% of TCP	
		TOS=0	DF=1	fragoff=0	options	res. flags	TCP	options	res. flags
BUF-1001462650	907036	96.26	88.50	100.00	0.00	0.00	90.35	2.89	0.00
BUF-1001473743	1026657	94.13	86.01	100.00	0.00	0.00	85.24	2.56	0.00
BUF-1001486582	809735	95.01	95.07	100.00	0.00	0.00	97.58	2.26	0.00
BUF-1001497670	375121	93.74	98.37	100.00	0.00	0.00	97.99	3.71	0.00
BUF-1001506918	348499	97.38	97.71	100.00	0.00	0.00	97.38	2.22	0.00
BUF-1001519802	660944	89.56	94.67	100.00	0.00	0.00	94.81	2.23	0.00
BUF-1001529100	1308250	85.47	94.93	100.00	0.00	0.00	94.28	1.75	0.00
BWY-1001462650	1424319	94.95	91.23	100.00	0.00	0.00	91.47	7.17	0.00
BWY-1001473743	1414983	95.89	87.23	98.71	0.00	0.00	86.91	11.87	0.00
BWY-1001486581	1115827	87.56	91.48	100.00	0.00	0.00	90.66	9.49	0.00
BWY-1001497671	763685	92.16	90.95	99.97	0.00	0.00	91.33	10.46	0.00
BWY-1001506918	699310	90.91	90.26	100.00	0.00	0.00	90.94	7.26	0.00
BWY-1001519803	1459321	97.28	92.36	100.00	0.00	0.00	92.46	19.03	0.00
BWY-1001529100	1324686	93.08	90.04	99.62	0.00	0.00	89.64	6.53	0.00
BWY-1001538343	1744003	90.68	92.18	100.00	0.00	0.00	91.69	17.81	0.00
COS-1001473743	867421	95.70	89.97	99.98	0.00	0.00	94.77	8.84	0.00
COS-1001486582	89833	95.55	92.75	99.91	0.00	0.00	96.83	5.51	0.00
COS-1001497671	1585481	97.03	92.50	99.89	0.00	0.00	97.93	5.37	0.00
COS-1001506917	1333572	92.84	93.64	99.99	0.00	0.00	98.74	10.33	0.00
COS-1001519803	2333952	94.68	86.96	99.49	0.00	0.00	94.57	8.07	0.00
COS-1001529101	2870711	97.01	85.12	99.57	0.00	0.00	95.30	8.24	0.00
COS-1001538342	2958614	95.60	88.99	99.38	0.00	0.00	95.05	9.48	0.00
MEM-1001473742	105478	97.88	92.92	100.00	0.00	0.00	94.15	4.44	0.00
MEM-1001486581	115441	98.86	97.07	100.00	0.00	0.00	97.10	2.87	0.00
MEM-1001497670	76412	98.30	95.74	100.00	0.00	0.00	96.59	2.02	0.00
MEM-1001506918	68729	98.32	95.71	100.00	0.00	0.00	96.38	3.48	0.00
MEM-1001519802	95266	95.65	96.04	100.00	0.00	0.00	96.75	4.12	0.00
MEM-1001529100	114926	98.44	97.40	100.00	0.00	0.00	97.95	3.10	0.00
MEM-1001538342	92195	95.01	95.23	100.00	0.00	0.00	96.23	7.03	0.00

Fig. 7. Redundancy in IP Traffic: Available header fields and properties. Each trace is about 90 seconds long and was collected on September 26, 2001.

packets currently use the reserved TCP header bits. However, TCP options are relatively rare, which suggests that NOP padding will not be very effective.

It is rare to find ACK=0. This is not surprising as TCP is a bidirectional protocol and acknowledgments for the symmetric channel can always be included “for free” when a packet is sent. However, in a LAN environment, ACKs are rarely delayed or lost. One could imagine a modification of the TCP stack for LANs in which acknowledgments are normally sent just once, and when the field is used for extra data in other situations. This would probably require too many code changes, however.

In the earlier set of traces, we did not study the prevalence of having the urgent flag set to zero. In the later traces we found that it is very rare ($<< 1\%$ of packets) for URG to be set. Hence the 16 bit urgent pointer field will almost always be available for use by additional data.

Most of our analysis of the traces has focused on the IP layer. Here we see some quite promising results. First, notice that in almost all cases the reserved TOS flags and the reserved IP flag are zero, and thus could be overloaded. Even better, about 90% of packets are not fragmentable (DF=1), which indicates that we can use the 13 bit fragmentation offset field and the 16 bit packet ID field. We notice that there are almost no IP options, which suggests that IP NOP padding will be ineffective.

As the traces do not show data link layer information, we can currently say nothing about the prospects for padding at the Ethernet level.

To summarize, we have identified in excess of 86 bits in various headers that could be used to communicate information in various cases. We studied our traces and considered the prospects for the bits in the IP header and the TCP header and found 54 bits that could be overloaded most of the time in practice, 22 at the TCP level and 32 at the IP level.

4 Proof of concept

Given the promising results of the previous section, we decided to test several of our IP layer mechanisms within a real network. To do so, we modified the IP module of Minet to inject information into outgoing packets and extract information from incoming packets. Minet [6] is a user-level TCP/IP stack that we have developed for educational purposes. It is not fast, but it is a compatible TCP/IP implementation that is very easy to modify. The information that we communicated was a random bit stream. About 220 new lines of C++ code were added to Minet, of which about 20 were involved with header editing, the remainder being debugging output. We evaluated using the following IP header fields: TOS, reserved, packet ID for don't fragment packets, and fragment offset for don't fragment packets. The total number of bits was 32.

We ran three experiments using Minet. In the first experiment, we communicated from Minet to a Linux 2.2 stack, checking what effect each of our mechanisms had on ordinary IP communication. The communication path traversed several managed and unmanaged layer 2 Ethernet switches. In the second experiment, we communicated from a Minet stack to another Minet stack with a layer 3 IP router (Cisco) in the communication path in addition to layer 2 Ethernet switches. Here we wanted to check to see how the router reacted to our "interesting" IP packets. We were out to verify that communication succeeded between the two stacks and that our embedded data was not overwritten by the router. In the third experiment, we communicated from a Minet stack to a Linux stack via the router. Here wanted to verify that IP communication was still possible.

Figure 8 shows the results of our experiments. Essentially, we found that using the fragment offset field was impossible, even for packets marked as not fragmentable. Both the Linux stack and the Cisco router rejected such packets. We also found that while using the reserved TOS bits did not affect IP communication, the router zeroed these bits and hence they could not be used to communicate additional information.

We were able to successfully use the reserved IP flag and the packet ID field to communicate additional information without affecting IP communication. We are relatively confident that these 17 bits can be used in other contexts as well.

Mechanism	Bits	Minet to Linux	Minet to Router to Minet	Minet to Router to Linux	Demon-strated bits
Reserved TOS bits	2	OK	FAILS	OK	0
Reserved IP flag	1	OK	OK	OK	1
Identifier when DF=1	16	OK	OK	OK	16
Fragment offset when DF=1	13	FAILS	FAILS	FAILS	0
NOP option padding	varies	untested	untested	untested	0
Total	**>=32**				**17**

Fig. 8. Proof of concept.

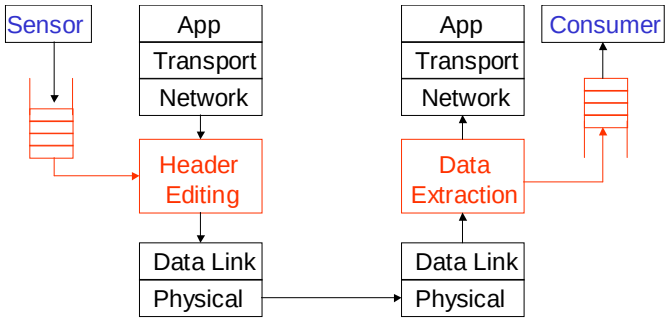

Fig. 9. Anticipated implementation.

In summary, then, we have shown it is possible to communicate an additional 17 bits of information on almost all IP packets through a typical networked environment, about $1/2$ of the number of bits we initially identified.

5 Disseminating dynamic resource information

The mechanisms we have described and studied provide an odd kind of communication channel. Like the channel formed by the IP protocol, packet loss, corruption, and reordering can occur. Unlike IP, however, the sender is also not in control over when or necessarily to whom data will be sent. What would this look like from the application level? How can we use this channel to communicate resource information?

The implementation that we anticipate is summarized in Figure 9. We envision that at the application level we would expose a send and receive queue, perhaps as files in the /proc filesystem on Linux or as a device driver represented by a /dev file. An application would enqueue messages consisting of bits and preferred IP addresses to which those bits should be sent, including wildcard addresses. In the normal course of operation, when a packet is about to be injected into the network, the stack would find the most relevant queued message and embed as much of it into the packet as possible into the

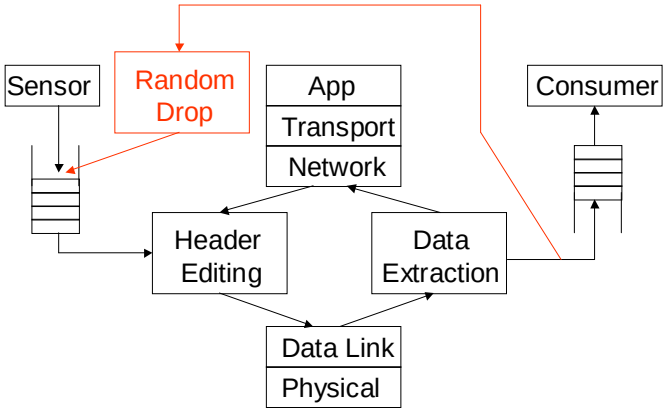

Fig. 10. Information diffusion concept.

packet's redundant fields. Any remaining data in the message would have to wait to hitch a ride on the next packet going to that destination. When a packet is extracted from the network, the bits would be stripped out of its redundant fields and queued with a source address and timestamp to be read by a user-level process. The timestamp would be determined without using packet bits simply through the local time and the TCP round-trip-time estimator and would only be available for TCP packets.

The 17 bits per packet that we have identified thus far would be sufficient to embed host load information, as typical entropy for a single host load measurement is on the order of 13-15 bits [5]. Longer messages would have to be encoded using forward error correction. Any such encoding or other transport-like functionality would be the responsibility of the user-level process.

One can also consider an interesting extreme use of our mechanism, illustrated in Figure 10. Instead of having the sender designate the destination, one could simply ignore the destination and choose the next message in the queue when injecting a packet into the network. Furthermore, some portion of incoming messages would be copied to the outgoing queue to be propagated further. In this way, messages would slowly diffuse outward from "close" hosts to more distant ones. The portion of messages that a host resends would control how far information would spread and thus the scalability of the scheme.

Security is a legitimate concern with these two models. Users choose the hosts which exchange data packets, but it is the owners of the hosts which determine what data is piggybacked onto those packets. The users must now trust the owners to a greater degree than before.

6 Conclusion

We have demonstrated the potential for disseminating dynamic resource information at low or no cost by exploiting the redundancy that exists in packet headers. The theoretical limits of that redundancy are quite high, and there also exist practical mechanisms

to exploit it whose prospects appear bright. Our simple proof-of-concept experiment demonstrated that it is possible to communicate an extra 17 bits of information on almost every IP packet traversing two common network paths, enough bits to usefully represent load or bandwidth information. Finally, we have described two models for how this new communication channel could be used.

The next steps in this work are to develop the kernel implementation discussed in the previous section, to test more of the redundancy exploitation mechanisms we pointed out, and to evaluate the mechanisms across a wider range of network environments. We are currently implementing the idea within the Linux kernel.

References

1. Internet protocol: Darpa internet program protocol specification. Internet RFC 791, September 1981.
2. Transmission control protocol: Darpa internet program protocol specification. Internet RFC 793, September 1981.
3. ABARBANEL, H. *Analysis of Observed Chaotic Data*. Institute for Nonlinear Science. Springer, 1996.
4. BYERS, J. W., LUBY, M., MITZENMACHER, M., AND REGE, A. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of ACM SIGCOMM 1998* (October 1998).
5. DINDA, P. A. The statistical properties of host load. *Scientific Programming 7*, 3,4 (1999). A version of this paper is also available as CMU Technical Report CMU-CS-TR-98-175. A much earlier version appears in LCR '98 and as CMU-CS-TR-98-143.
6. DINDA, P. A. The minet tcp/ip stack. Tech. Rep. NWU-CS-02-8, Northwestern University Department of Computer Science, January 2002.
7. DINDA, P. A., AND O'HALLARON, D. R. An extensible toolkit for resource prediction in distributed systems. Tech. Rep. CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July 1999.
8. JACOBSON, V. Compressing tcp/ip headers for low-speed serial links. Internet RFC 1144, February 1990.
9. LOWEKAMP, B., MILLER, N., SUTHERLAND, D., GROSS, T., STEENKISTE, P., AND SUBHLOK, J. A resource monitoring system for network-aware applications. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC)* (July 1998), IEEE, pp. 189–196.
10. NATIONAL LABORATORY FOR APPLIED NETWORKING RESEARCH. Nlanr network analysis infrastructure. http://moat.nlanr.net. NLANR PMA and AMP datasets are provided by the Natioanl Laboratory for Applied Networking Research under NSF Cooperative Agreement ANI-9807579.
11. SHANNON, C. E. A mathematical theory of communication. *Bell System Tech. J. 27* (1948), 379–423, 623–656.
12. SKICEWICZ, J., DINDA, P., AND SCHOPF, J. Multi-resolution resource behavior queries using wavelets. In *Proceedings of the 10th IEEE Symposium on High-Performance Distributed Computing (HPDC 2001)* (August 2001), pp. 395–405.
13. TIERNEY, B., AYDT, R., GUNTER, D., SMITH, W., TAYLOR, V., WOLSKI, R., AND SWANY, M. A grid monitoring architecture. Tech. Rep. GWD-PERF-16-2, Global Grid Forum, January 2002.

14. WOLSKI, R. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th High-Performance Distributed Computing Conference (HPDC97)* (August 1997), pp. 316–325. extended version available as UCSD Technical Report TR-CS96-494.
15. ZINKY, J., BAKKEN, D., KRISHNASWAMY, V., AND AHAMAD, M. Pass — a service for efficient large scale dissemination of time varying data using corba. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS 99)* (1999).