# Netzwerktests

- Betreuer: Priv. Doz. Dr. habil. Thomas Kropf
- Vortrag: Horst Rechner

Eberhard Karls Universität Tübingen

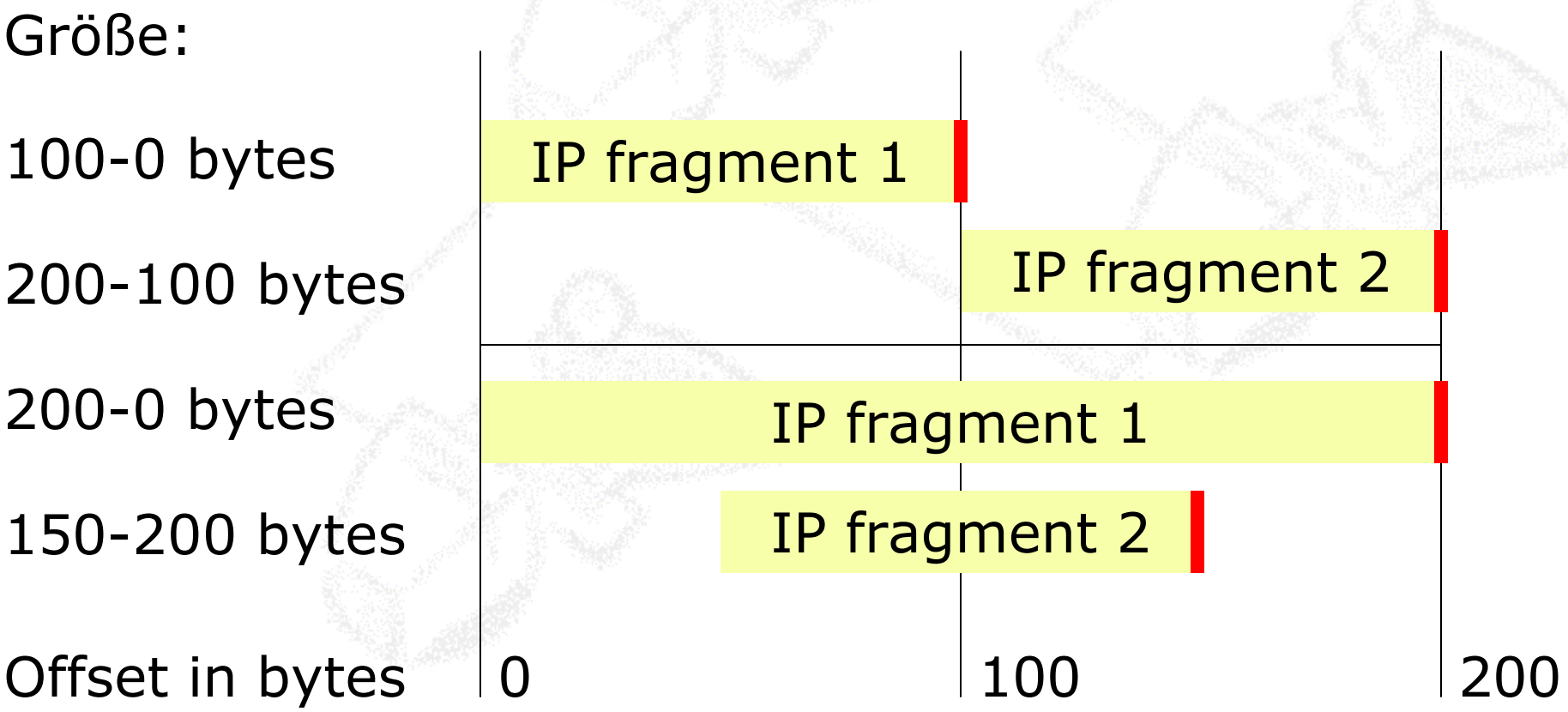
Wilhelm-Schickard-Institut für Informatik
Arbeitsbereich Technische Informatik

Seminar Pleiten, Pech und Pannen in der Informatik

07. Februar 2002 (Wintersemester 2001/2002)



# Teardrop DoS 1998



# Kosten?

- 3 Typen von Denial of Service:
  - Software
  - Flooding (SYN Flood)
  - DDOS
- Erste wissenschaftliche Studie:
  Internet Denial-of-Service Activity
  University of California, San Diego
  USENIX Security ’01
- 12805 Attacken in 3 Wochen

Quellen: [Moore01]

# Kosten?

- Kosten schwer zu kalkulieren

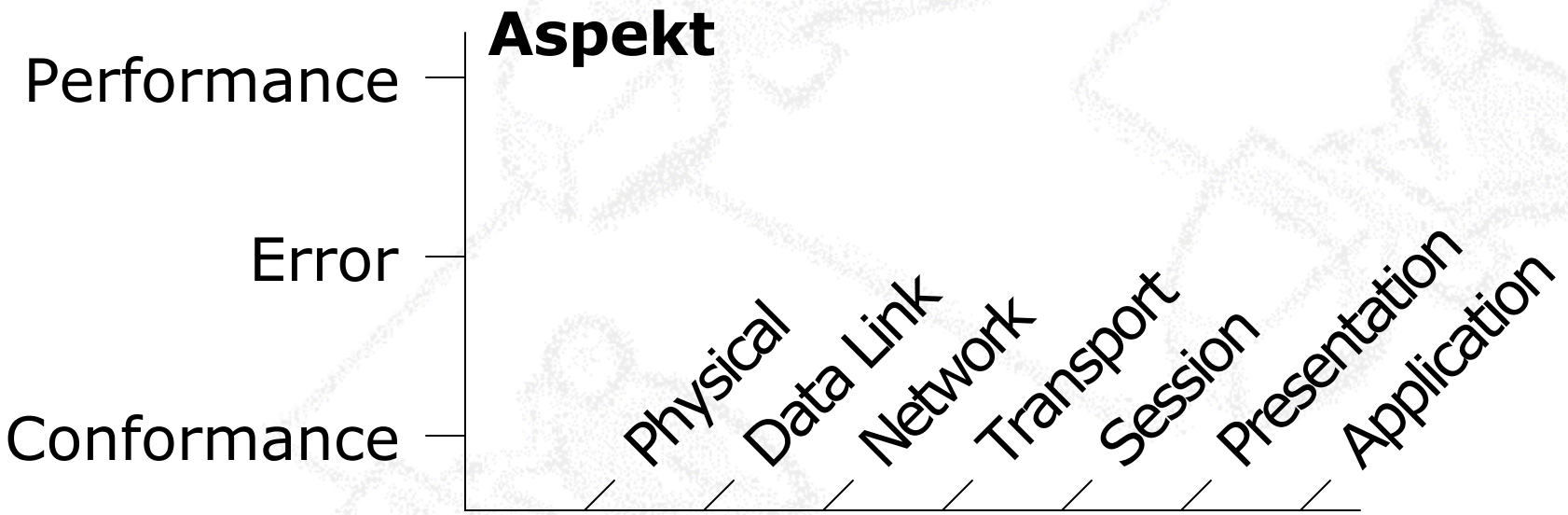
- CSI/FBI Umfrage:
- 2001 - 378 Millionen $ (186 Befragte)
- 2000 - 276 Millionen $ (249 Befragte)

- Viele Organisationen erteilen keine Auskunft oder haben überhaupt keine Kenntnis

# Welchen Aspekt?

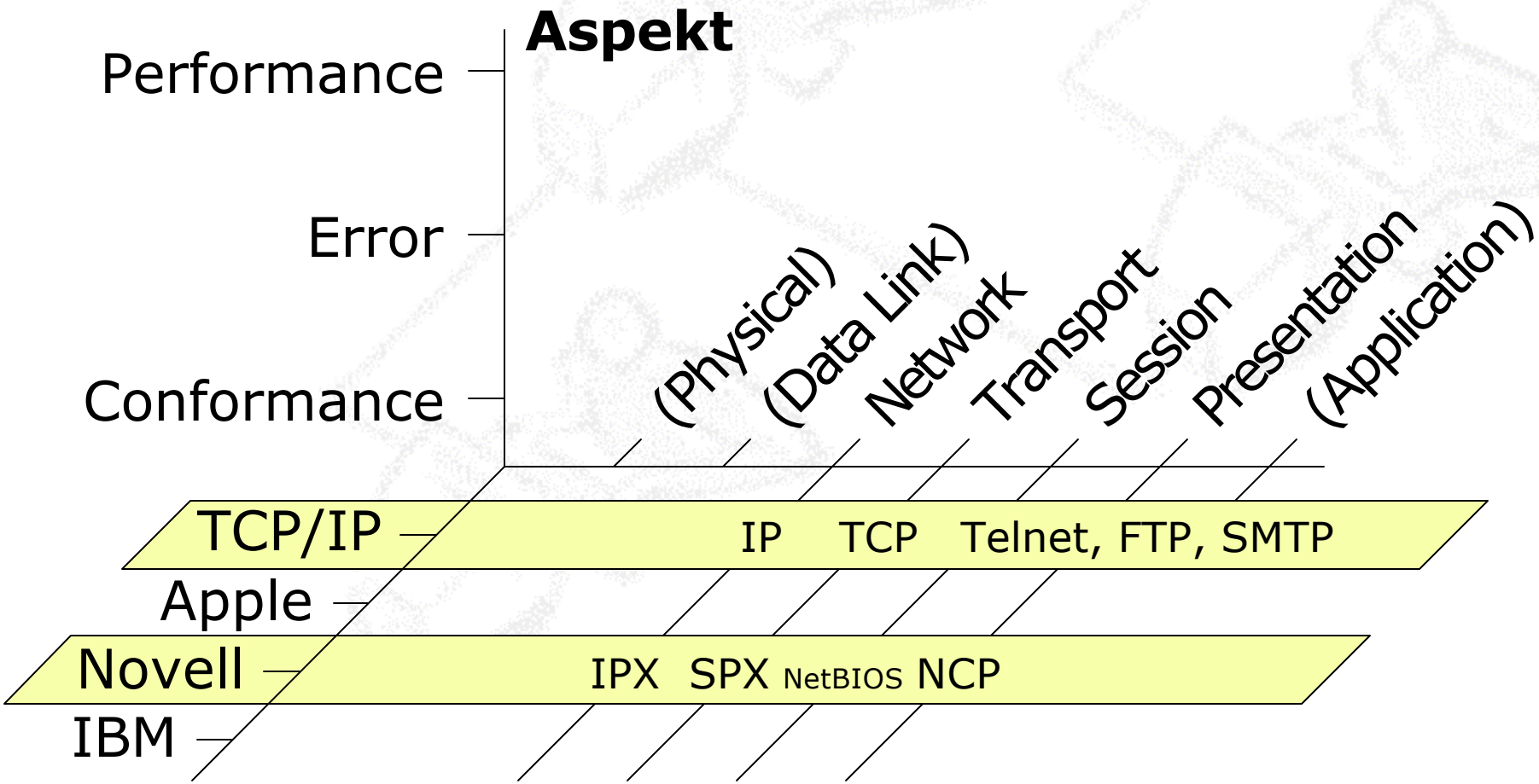
| | |
|---|---|
| Performance | (Stress / Load) |
| Error | (Security) |
| Conformance | (Compliance / Functionality) |

# Welcher Layer?



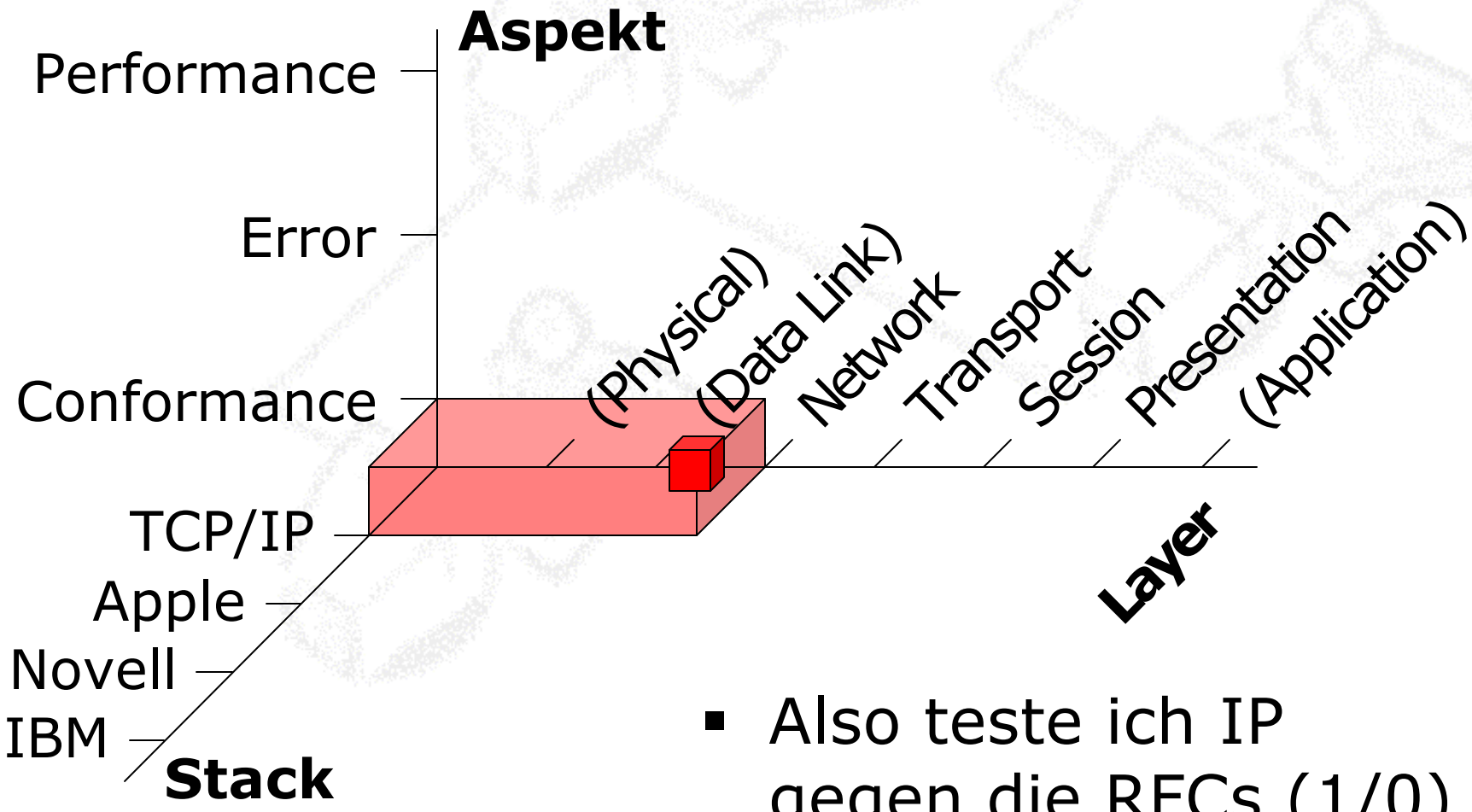

Quellen: [Lex], [Lewis97]

# Welcher Stack?



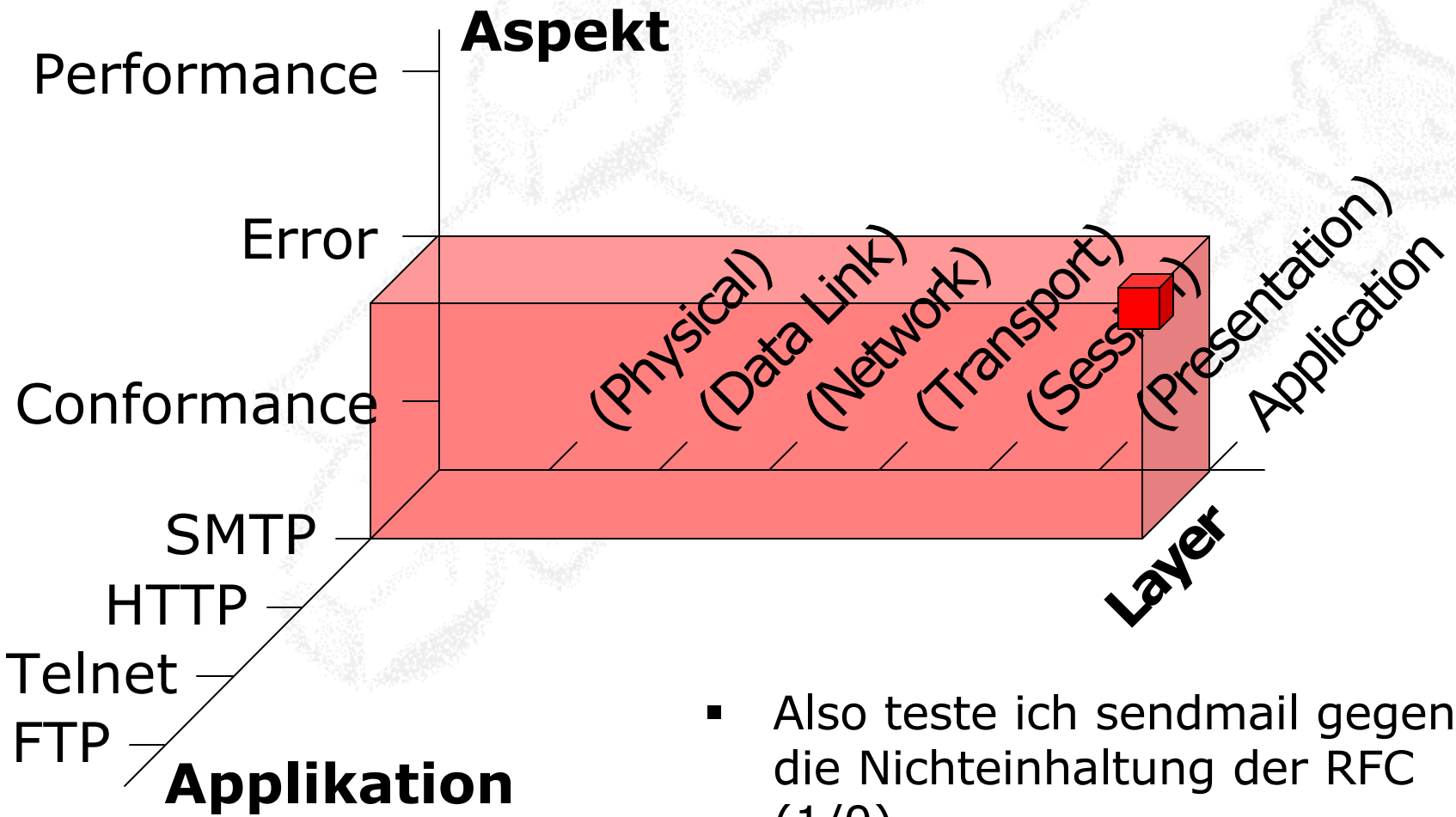

Quellen: [Lex]

# Wir bekommen ein Tripel




- Also teste ich IP gegen die RFCs (1/0)

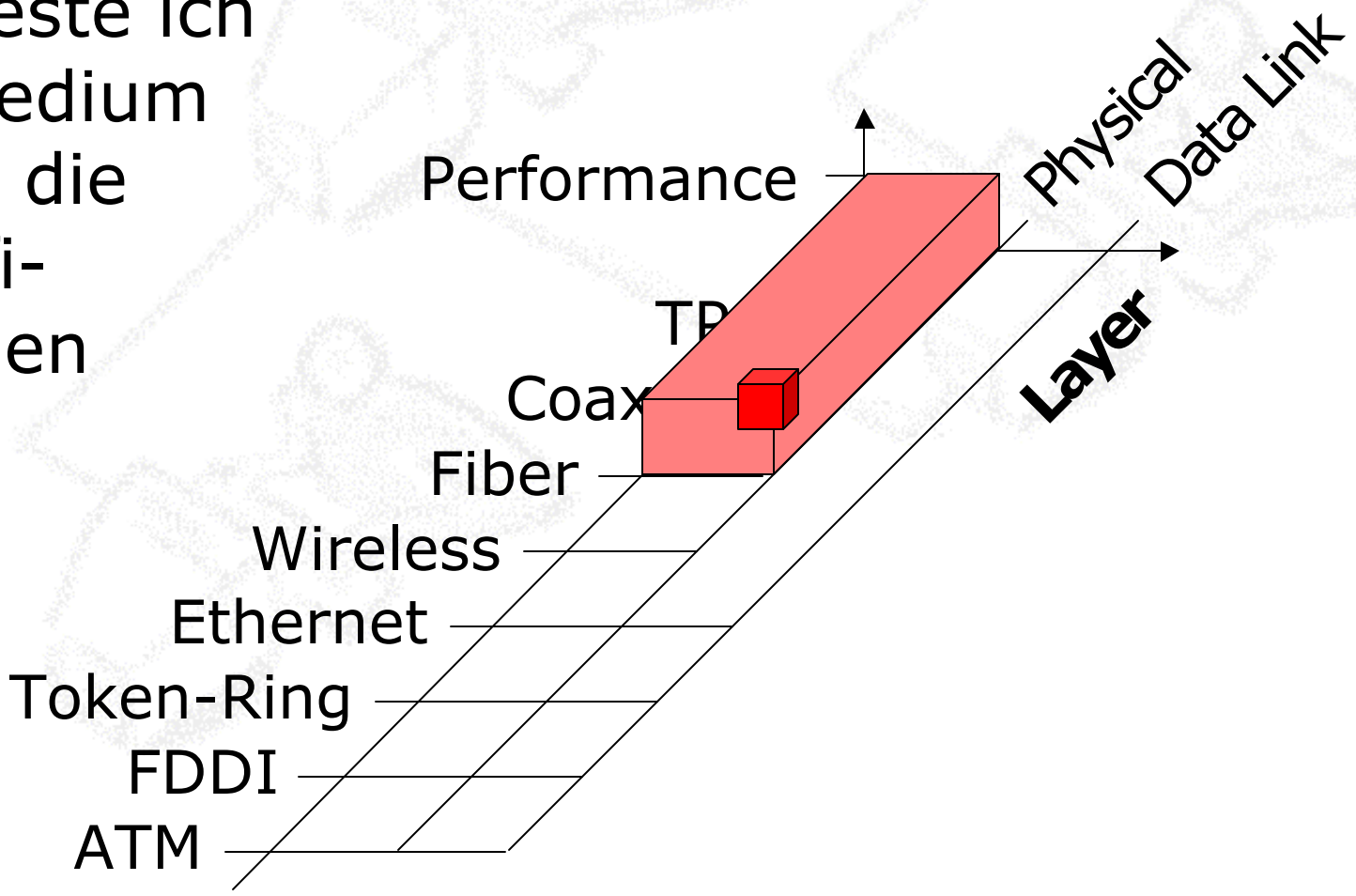
# Was ist mit Layern 1, 2 und 7?




- Also teste ich sendmail gegen die Nichteinhaltung der RFC (1/0)

# Was ist mit Layern 1, 2 und 7?

- Also teste ich das Medium gegen die Spezifi-kationen



# Teardrop DoS 1998

- Tripel: Error, Network, TCP/IP
- Aus IETF RFC 760:

„The fragment offset field of the second new internet datagram is set to the value of that field [the offset field] in the long datagram [the unfragmented one] plus NFB [the number of fragment blocks].“

200-0 bytes

IP fragment 1

150-200 bytes

IP fragment 2

# Was hätte hier helfen können?

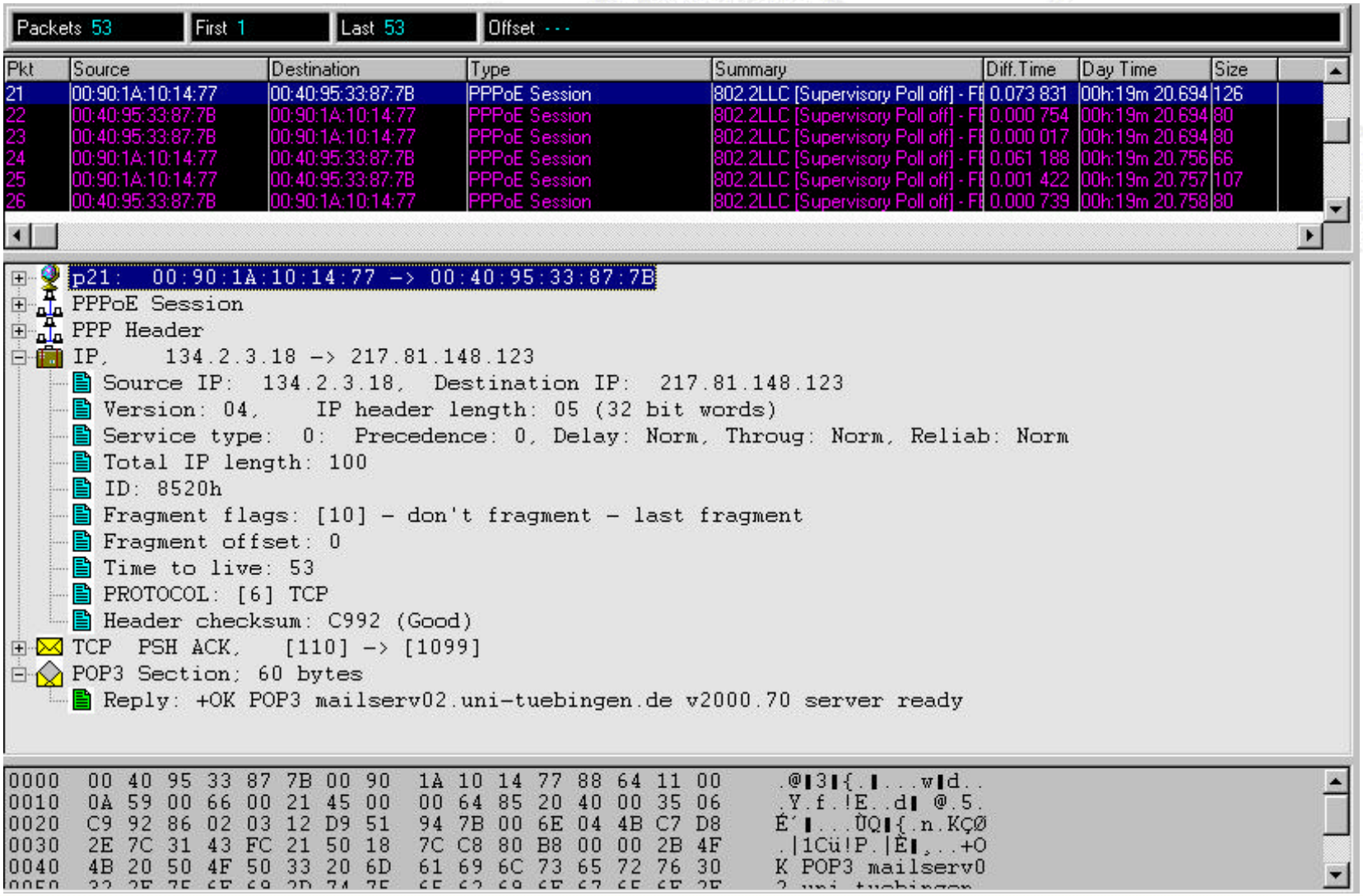
- Test der Implementation gegen die RFCs (Conformance)
- Test gegen Abweichungen von den RFCs (Error)

> Packet Generator / Analyzer

# Empirix ANVL

- **IP Test Suites**
  IP RIP (v1 and v2) Gateway / OSPFv2 (RFC1583/2328) / BGP4 (RFC1771) / RMON (RFC1757/RFC1513)
- **PPP Test Suites**
  Basic PPP (with tests for LCP, PAP, and CHAP) / IPCP (RFC1332) / Multilink PPP (RFC1717/RFC1990) / VJ Test Suite (TC/IP, RFC1144) / Spanning Tree (IEEE 802.1d)
- **Multicasting Test Suites**
  IGMP (RFC2236v2) / DVMRP (IETF Draft 3) / PIM (sold as one unit) / Sparse Mode - IETF Draft#1v2 / Dense Mode - IETF Draft#3v2
- **TCP Test Suites**
  Core (RFC 793, 1122) / Advanced (RFC 2001, 2581, 1191, 2385) / Performance (RFC 1323, 2018)
- **VPN Test Suites**
  PPTP (IETF Draft 2) / L2TP (RFC2661) / IPSec AH (RFC2402/2401) / IPSec ESP (RFC2406) / IPSec IKE (RFC2409/2408) / L2TPSec (RFC2661)

# Packetanalyzer



# Referenzwerte

Für jeden Test braucht man Referenzwerte.

Wie findet man diese Werte?

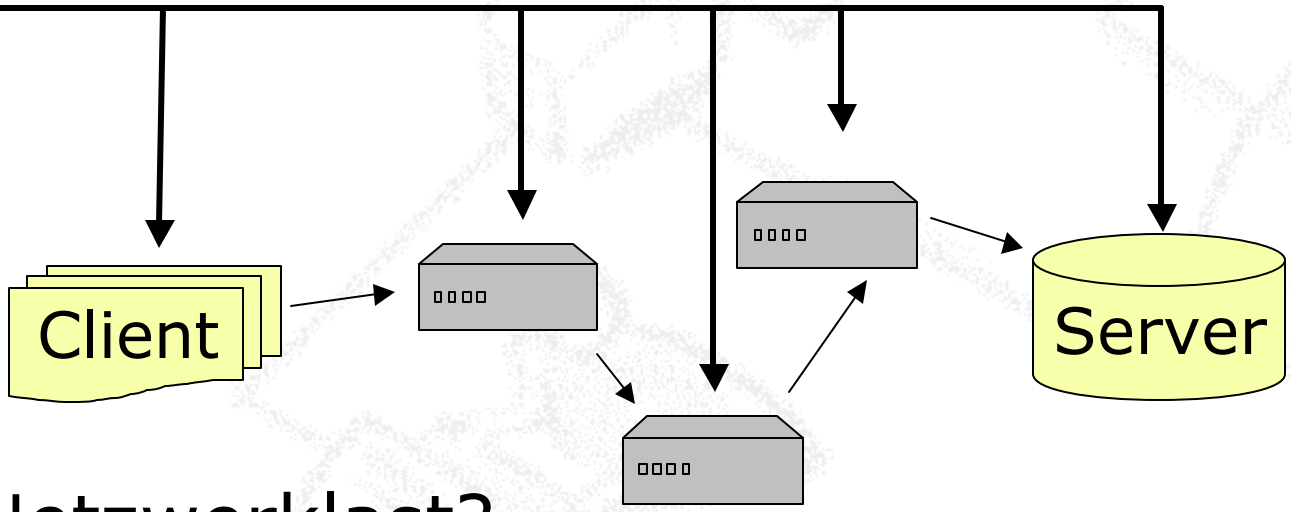
# Performance

- „Meine Email ist langsam !“

> Tripel: Performance, Application, SMTP

# Netz

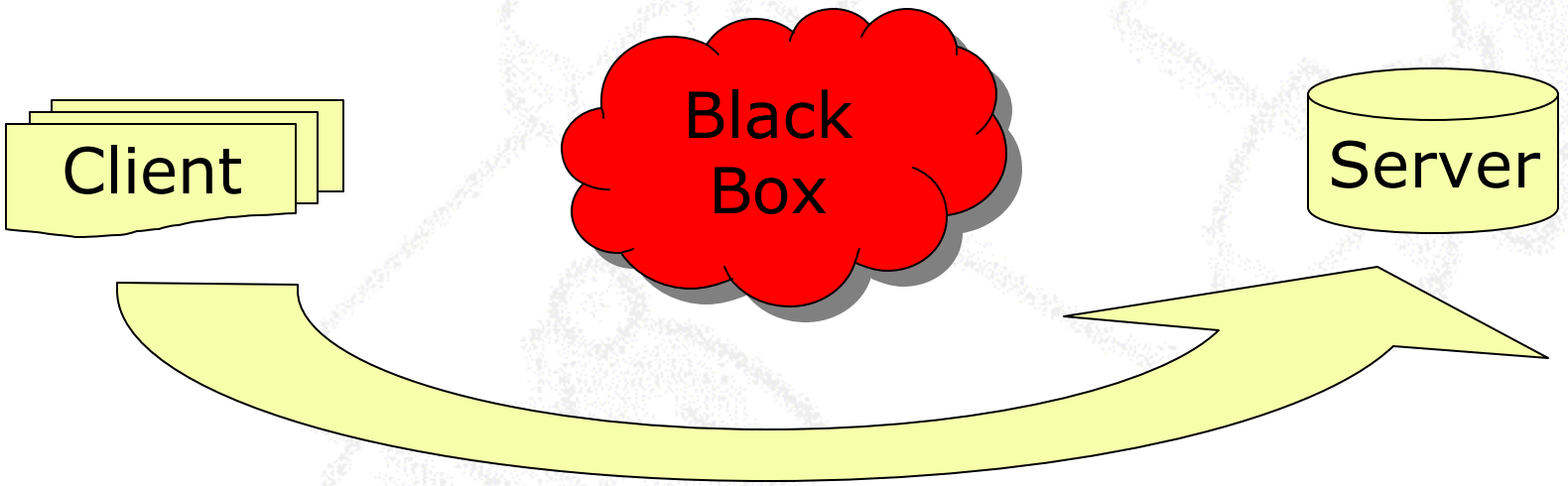
- Protocol Analyzer: Timestamp




- Netzwerklast?
- Kolllisionen?
- Retransfers?
- Route (PPS / Latenz der Komponenten)?

# Client und Server

- Protocol Analyzer: Timestamp




- Think Time?
- Serverlast?

# Ausblick

- Weitere Anforderungen an Netzwerktests

- Performance oder QoS: VoIP
  - Verfügbarkeit und garantierter Durchsatz
    z.B. bei G.723.1 und 6.4 kbps (UDP: 13.3 kbps)

  > Priorität der Pakete bei IPv6

  - geringe Latenz (< 250 ms) und Änderung der Lantenz (Jitter)
  - Hardware
    (vgl. TI TMS320C5000 200 MHz und 400 MIPS)
    Encoder braucht ca. 20 MIPS
    Decoder braucht ca. 2 MIPS

# Quellen


- [Acter02]
  White Papers
  Acterna, LLC
  http://www.acterna.com/technical_resources/white_papers/index.html
- [Burg00]
  Week 15: TCP/IP security
  Prof. Mark Burgess, University College Oslo, Faculty of Engineering, Norway,
  http://www.iu.hio.no/~mark/lectures/sysadm/html/SA15.eng.html
- [CPTIT]
  -
  Corporate Persuasion Through Internet Terrorism
  http://63.105.33.158/security/denial/w/teardrop.dos.html
- [Emp02]
  ANVL™ Automated Network Validation Library
  Empirix
  http://www.empirix.com/empirix/voice+network+test/products/_anvl+automated+network+test.html?page=new_home&link=dropdown_anvl
- [Evans02]
  Notes on Texas Instruments Processors
  Brian L. Evans, The University of Texas, Austin
  http://www.ece.utexas.edu/~bevans/courses/realtime/lectures/23_DSPs/texasInstruments.html

# Quellen


- [IETF80]
  DoD Standard Internet Protocol
  Information Sciences Institute, University of Southern California, 1980
  http://www.ietf.org/rfc/rfc0760.txt
- [Lewis97]
  James Bond Meets The 7 Layer OSI Model, 1997
  Richard Lewis,
  http://www.pe.net/~rlewis/Resources/james.html
- [Lex]
  Protocol Stacks in Relationship to the OSI Model
  Lexicon Consulting
  http://www.lex-con.com/osimodel.htm
- [Moore01]
  Inferring Internet Denial-of-Service Activity
  David Moore, Geoffrey Voelker und Stefan Savage, CAIDA University of California, San Diego
  http://www.caida.org/outreach/papers/2001/BackScatter/

# White Papers

## Communications Technology

- 2.048 Mbps Technology Basics and Testing Fundamentals (File: 598.4KB)
- ADSL Basics (DMT) (File: 291.9KB)
- An Introduction to the Essensials of ISDN (part I)
- ATM Support for Voice Applications (278K)
- "DECT--Technology on the Road to Success"
- Frame Relay Clears the Hurdles
- Frame Relay Flow Control and Data Transmission
  - Part 1: Basic Frame Relay Transmission (147K)
  - Part 2: TCP Over Frame Relay (107K)
  - Part 3: TFTP Over Frame Relay (61.3K)
- *GSM Pocket Guide*
- HDSL Basics (File: 288.9KB)
- "How to Improve the Quality of Service of ISDN Networks while Minimizing Maintenance Costs" NEW!
- Implementing a Distributed Test Solution for Wide Area Digital Transmission Facilities (File: 406.1KB) NEW!
- ISDN, Basic Rate and Primary Rate Tutorial (3.02 MB)
- *ISDN Test Solutions: Access and Equipment* *Edition 1*
- *ISDN Supplementary Services, ISDN Pocket Guide No. 1* *Edition 3*
- *MPEG-2 Pocket Guide* *Edition 1* NEW!
- "Perspectives on TMN"
- *PPP Troubleshooting, ISDN Pocket Guide No. 2* *Edition 3*
- *SDH Pocket Guide*
- *SONET Pocket Guide*
- Straight Answers about ATM Testing
- T1 Basics (File: 355.1KB)
- Tandem Connection Monitoring - Who has caused the impairment? NEW!
- *Test Solutions for Digital Networks* (book)
- Testing Beyond The Physical Layer
- The Fundamentals of DS3 (File: 376.1KB)
- The Fundamentals of SDH (File: 131.3KB)
- The Fundamentals of SONET (File: 161.1KB)

## Data Network Analysis

- Always On / Dynamic ISDN (AO/DI) Turn-up and Maintenance
- "Analyzing the Analyzer: The Essential Features for Protocol Analysis"
- "ATM Testing Challenges"
- "Benchmarking Methodology for Ethernet Switches"
- "How to Handle Test Access in Switched Ethernet Environments"
- *ISDN PPP Troubleshooting Guide, ISDN Pocket Guide No. 2*
- "Network Applications: Are They Performing?" Part 1: A Client's Perspective
- "Network Applications: Are They Performing?" Part 2: The Server vs the Network "
- "New Standards for Network Analysis: Expert Systems Come of Age" (Net3 Group white paper)
- Selecting Network Analysis Tools for Network Service and Support Organizations
- "Troubleshooting Complex Network Problems with Advanced Network Analysis Equipment" (Tolly Group white paper)
- What is a Protocol Analyzer?

**Data Network Baselining**

- " How to Choose a Baselining Tool"
- *LAN Troubleshooting and Baselining* (Pocket Guide)
- "Network Baselining, Part I: Understanding the Past to Predict the Future"
- "Network Baselining, Part II: The Big Picture"
- "Network Baselining, Part III: Focus on the Node"

# "Network Applications: Are They Performing?"

## Part 1: A Client's Perspective

a white paper by Alan Chapman

Network managers hear it every day: "Why is the network so slow?"

In the first of two white papers that examine the performance of network applications, Alan Chapman, manager of WGUSA's Professional Services group, looks at the issue from the user's perspective.

The second installment considers server- and network-induced performance issues.

**Adobe PDF Format**

complete text as a PDF file (49K)

# Embedded Secure Document

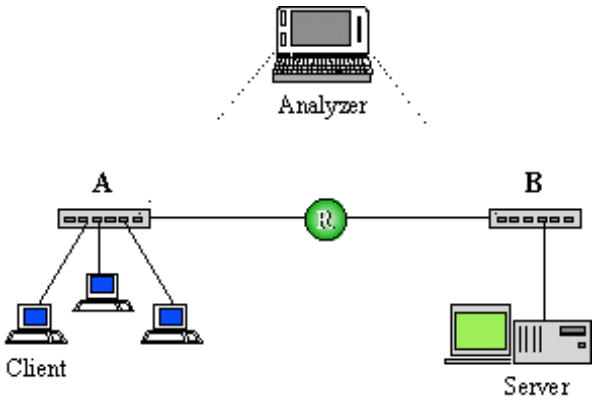
The file *http://www.acterna.com/downloads/white_papers/telecom_and_datacom/netapps1_wp.pdf* is a secure document that has been embedded in this document. Double click the pushpin to view.

# "Network Applications: Are They Performing?"

## Part 2: The Server vs the Network

a white paper by Alan Chapman

"Why is the network so slow?" Network managers hear it every day.




In the second of two white papers that examine the performance of network applications, Alan Chapman, manager of WGUSA's Professional Services group, considers server- and network-induced performance issues.

The first installment looked at application performance from the client perspective.

**Adobe PDF Format**

complete text as a PDF file (29K)

# Embedded Secure Document

The file *http://www.acterna.com/downloads/white_papers/telecom_and_datacom/netapps2_wp.pdf* is a secure document that has been embedded in this document. Double click the pushpin to view.

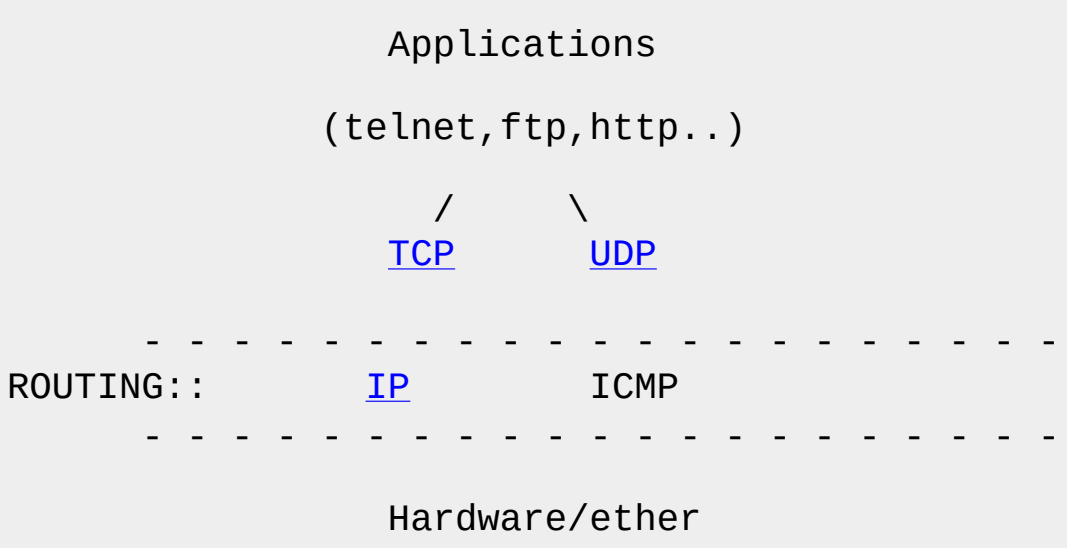
# Week 15: TCP/IP security

**Fact of the week**

Open protocols, transmitted as plain text are a problem for security. With ever greater numbers of `smart' products on the market that makes security all the more important. Early mobile phones could be tapped from passing cars. The thieves could capture their codes and program new phones with these. They would then sell these on the black-market so that others could phone on the bill of innocent passers-by. Increasingly companies are using cheap offshore programming help in countries like India, Hungary, Ireland, Israel and Singapore. All this makes the security of remote services extra important.

**IPV4**

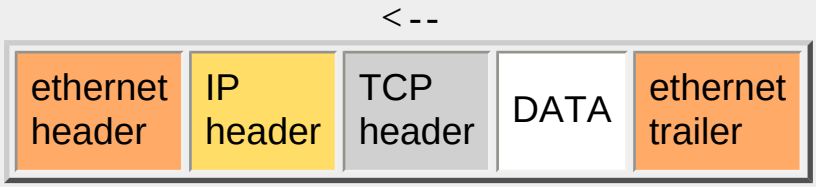
The internet protocol was conceived as a military project in the '70's. The aim was to produce a *routable* network protocol. The version of this protocol in use today is version 4, with a few patches. Let's revise some of the basics of IPV4, which we discussed earlier in the operating systems course. TCP/IP is a transmission protocol which builds on lower level protocols like ethernet and gives it extra features like `streams' or virtual circuits, with automatic handshaking. UDP is a cheaper version of this protocol which is used for services which do not require connection-based communication. The TCP/IP protocol stack consists of several layers:




At the application level we have text-based protocols like telnet and FTP etc. Under these lies the TCP (Transmission Control Protocol) which provides reliable connection based handshaking, in a virtual circuit. TCP and UDP introduce the concept the *port* and the *socket* (=port+IP address). We base our communications on these, and thus we also base the security of our communications on these. Under TCP/UDP is the IP transport layer, then ethernet or token ring etc. ICMP is a small protocol which is used by network hardware to

send control and error messages as a part of the IP protocol set. e.g. ping uses ICMP

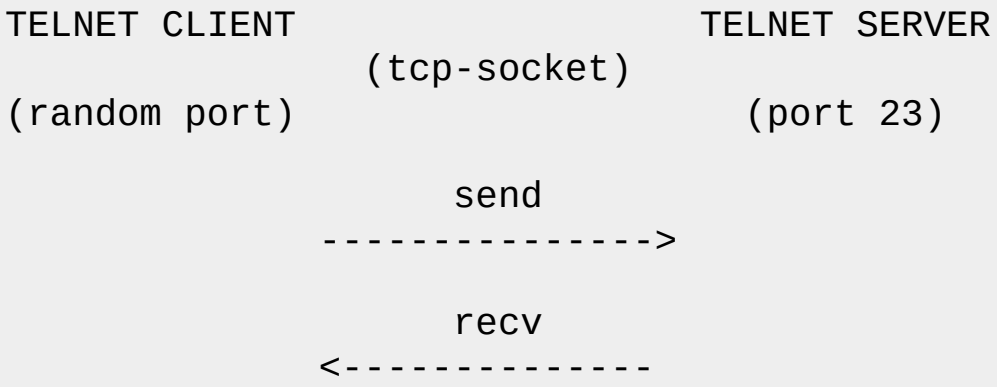
With all of its encapsulation packaging, a TCP/IP packet looks like this:




Header structure is added for each level of the protocol creating packets of the form shown above. In addition to this form of packaging, special packets can be packed several times in order to perform efficient distribution. For instance, the MBONE (multicast backbone) network spreads encapsulated multicast packets to local networks without having to send multiple packets. They are first sent to a multicast router and there unpacked, revealing the addresses of local hosts which should receive the packets.

Network crackers have been clever in exploiting problems in the design and implementation of TCP/IP for their own purposes. In order to protect ourselves against the problems which these people cause, we have to understand a few things about how the TCP/IP protocol works. This is fairly complicated, but we can summarize the important elements.

Let us consider telnet as an example and see how the telnet connection looks at the TCP level.




Telnet opens a socket from a random port address (e.g. 54657) to a standard well-known port (23) where the telnet service lives. The combination of a port number at an IP address, over a communication channel is called a socket. The only security in the telnet service lies in the fact that port 23 is a reserved port which only root can use. (Ports 0-1023 are reserved).

**Filtering routers**

Modern routers are programmable devices which allow ACLs (Access Control

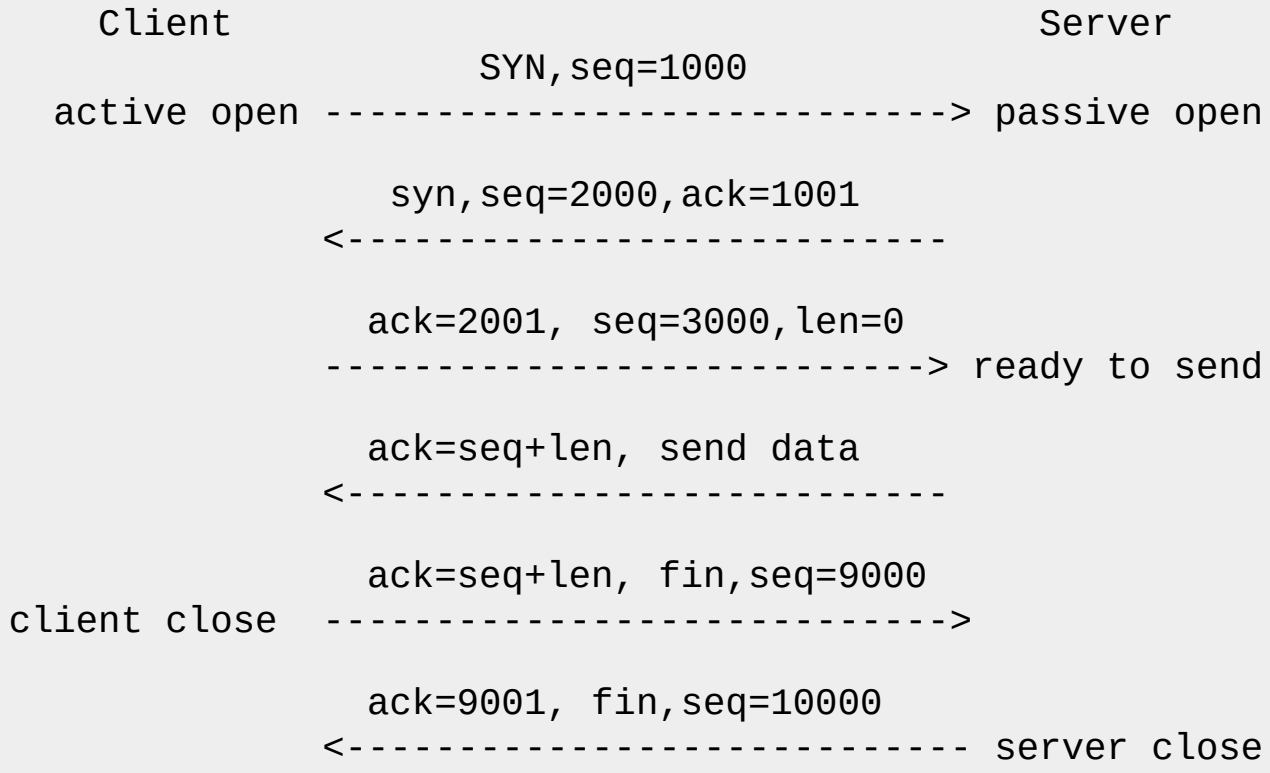
Lists) and access control through tables. We can make rules which determine who is allowed to send protocols through the router. Filtration can be based on various criteria:

- Protocol type (TCP/UDP/ICMP)
- Whether traffic is incoming or outgoing
- IP address (Source or destination)
- Port number (Source or destination)
- SYN flag in the TCP header (see below)

To understand more about this we need to revise some basics about TCP/IP.

**TCP circuits**

The TCP protocol guarantees to deliver data to their destination in the right order, without losing anything. In order to do this it breaks up a message into segments and numbers the parts of the message according to a sequence. It then confirms that every part of that sequence has been received. If no confirmation of receipt is received, the source retransmits the data after a timeout. The TCP header contains handshaking bits. Reliable delivery is achieved through a three-way handshake. Host A begins by sending host B a packet with a SYN (synchronize) bit set and a sequence number. This provides a starting reference for the sequence of communication. Host B replies to this message with a SYN,ACK which confirms receipt of an open-connection request and provides a new sequence number which confirms identity. Host A acknowledges this. Then B replies with actual data. We can see this in an actual example.



```
                    ack=10001
                  ----------------------------> connection closed
```

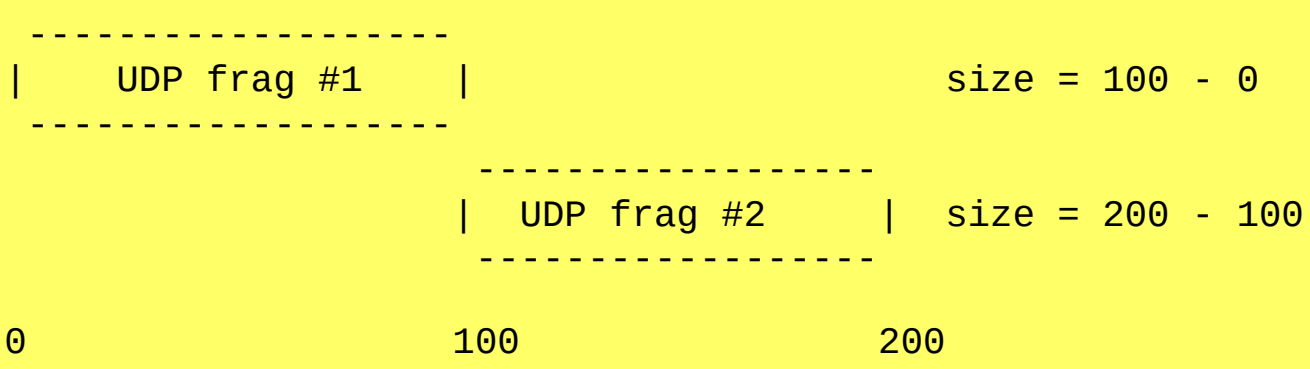
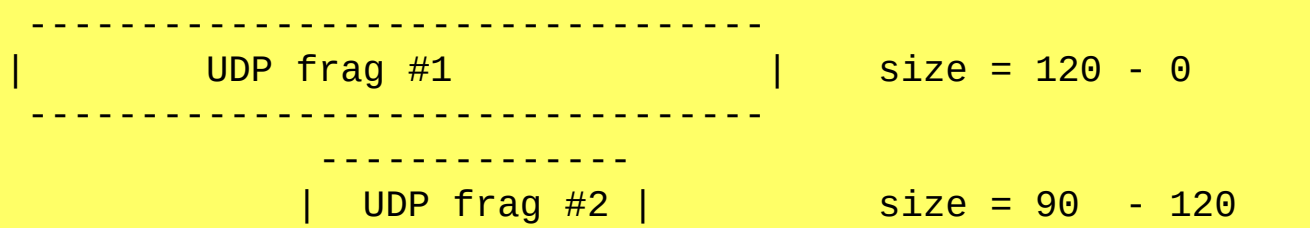
This handshaking method of sending sequence numbers with acknowledgement allows the TCP protocol to guarantee and order every piece of a transmission. The ACK return values are incremented by one because in earlier implementations this would be the next packet required in the sequence. This predictability in the sequence is unfortunately a weakness which can be exploited by so-called *sequence guessing* attacks. Today, in modern implementations, sequences numbers are randomized to avoid this form of attack. Older operating systems still suffer from this problem. Future implementations of TCP/IP will be able to solve this problem by obscuring the sequence numbers entirely through encryption.

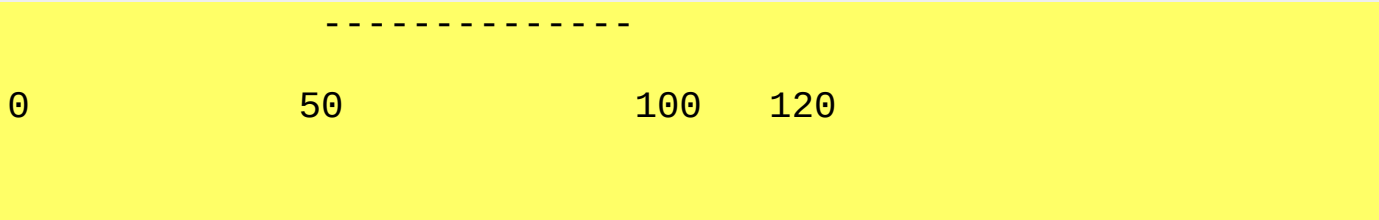
The TCP handshake is useful for filtering traffic at the router level, since it gives us something concrete to latch onto. TCP would rather drop a connection than break one of its promises about data integrity, so if we want to block telnet connections, say, we only have to break one part of this fragile loop. The usual strategy is to filter all incoming connections which do not have their ACK bit set, using router filtering rules. This will prevent any new connections from being established with the `outside'. We can, on the other hand, allow packets which comes from inside the local network. This provides a simple router-level firewall protection. It is useful for stopping IP spoofing attempts. The UDP protocol does not have SYN,ACK bits and so it is more difficult to filter.

**IP spoofing**

The cleverest system crackers are those who can fake IP packets. The idea is that those who can manage to spoof an IP packet can make it appear to come from a different host, preferably one which has special privileges. This can obviously be used to circumvent access control mechanisms.

There are two ways we can do this. The first is at the **application level**. Some programs restrict access on the basis of the hosts packets come from. By providing the identity of another machine it might be possible to trick the access controls. This can be checked fairly easily by cross checking that the caller is who they actually claim to be. Double, reverse DNS lookup is a typical strategy. Also a direct connection to the claimed caller is possible. The `ident` service (pidentd) can be used to verify the identity of a caller at the TCP level, if the caller is running the service (it is not standard). A third possibility is to use a password or some other shared secret. The other possibility is to use the **protocol level**.

| | |
|---|---|
| TCP | A clever sender can forge IP addresses in the IP header. In these cases TCP connections can be stopped by a filtering router. For example. we can block IP addresses which appear to come from inside a firewall but actually come from outside (these must be forgeries). Another type of attack is *sequence guessing*. Here a TCP circuit can be broken in the middle by address forging and by guessing the simple sequence numbers of a TCP circuit, in order to hijack the connection (often combined with an attack which takes down the true machine). Newer implementations randomize sequence numbers to avoid this. Another form of TCP attack is SYN-flooding. This is a Denial of Service attack prevents a host from using TCP connections. Host A sends host B a large number of SYN packets (open connection) which appear to come from the address of host C which is not on-line. Since host C cannot respond with an ACK, these connections build up (until they time-out) thus filling up the TCP table. Once this is full, the host can no longer receive more TCP communication. It is important to take host C off-line, otherwise the forged address would be detected and host C would send and ICMP reset to say that it did not initiate the connection itself. This sort of attack could be stopped if all the world's routers refused to send packets with forged *source-addresses*. |
| UDP | The UDP protocol is more primitive than TCP and is therefore easier to spoof. A special type of attack which is now famous for creating the 12 hour attack on NT machines in 1998 is called Teardrop. In a Teardrop attack, UDP fragmentation is used to create a kernel panic. Fragmentation normally occurs when packets are routed. A packet can be fragmented in order to optimize transfer conditions to a particular network layer protocol. Normally this will look like this:<br><br>  <br><br><br>The fragments are always reconstructed at their final destination. The size of the fragments (in faulty implementations) is worked out by subtracting the previous offset from the new one. This value is used to allocate memory for the fragment at destination. In a Teardrop attack, the fragments are forged and made to look like this:<br><br>  <br> |

| | |
|---|---|
| |  <br>The final packet is calculated to have a negative size. The destination host tries to allocate -40 bytes and this crashes the kernel. |
| ICMP | Another protocol which works in parallel with IP is the ICMP control protocol. This is used to send control messages (like reset) and error messages between the network hardware. It can also be used for Denial of Service attacks. External routers can safely block many ICMP packets, like ping. The Smurf attack is an ICMP Denial of Service attack. |

**Network forensics and intrusion detection**

A new technology which is starting to emerge is that of Intrusion Detection. The idea is to detect attempts at network attack, as they happen, or in the final instance afterwards. See, for instance, Network Flight Recorder. By looking at every packet on the network it is hoped to see suspicious looking activity, port scanning attempts and bad protocol usage. This requires a lot of resources (disk and CPU) and it has two fundamental hindrances:

- **Fragmentation**: Packets which get fragmented are only reconstructed at the end destination. If a suspicious packet is fragmented in an unfortunate way, pattern matching algorithms will not be able to see the bad stuff. A clever attacker could always arrange for spoofed packets to be fragmented at source.
- **Switching/Routing**: Switches and routers limit the spread of traffic to specific cables. An intrusion detection system needs to see all packets in order to cover every attack.

**Password sniffing (telnet/ftp)**

Many communications standards were introduced before it was possible for normal users to have control of their own computers. Any security in these protocols was based on the fact that normal users would never have administrator (root) privileges. Telnet and ftp are examples of this. These programs send passwords in plain, unencrypted text, for anyone to see. To prevent this it is possible to use a system of one-time passwords. This has been adopted by many banks offering Internet banking.

One-time (disposable) passwords are passwords which are valid only once. If an intruder manages to read a one-time password by tapping the network, then it is of no use, since it is invalid as soon as it has been used. The idea is not unlike the idea behind TCP sequence numbers. The point behind this scheme is to use a private password to generate a sequence of throw-away passwords. As long as both sender and destination hosts know the private passwords, they can use it to encrypt and decrypt a random string. The string must be

randomized or sequenced to avoid replay attacks, but the main point is that it is not the password itself. A string encrypted with the true password will only be decrypted by the true password, so it is possible to verify that both sender and destination agree on the password without ever having to send it over the network.

There are various systems which use this technique, e.g. MIT's kerberos. Here is one of the original examples from AT&T:

S/KEY one-time passwords

| | |
|---|---|
| You wish to establish a connection between host A and host B | You have previously set a password on host B. |
| You telnet to host B | The Login prompts you with an encryption string: 659 ta55095 |
| You use "659 ta55095" pluss your own password to create a key on host A: | `key 659 ta55095`<br>`password: passord på B`<br>`EASE FREY WRY NUN ANTE POT` |
| You type in your one-time password for telnet<br><br>`"EASE FREY WRY NUN ANTE POT"` | Access granted. |

The peculiar string generated by this process is meant to be easy to type in. Today newer systems can be built which do all of this behind the scenes. The advantage of this system is that no real secrets are ever sent over the network. Instead we make used of a shared secret to send mutually understood data over the net.

# Thought of the week

> Many of the forms of attack above would be impossible if the underlying IP packets were private (encrypted), ie. if they could not be read by everyone.

---

Mark Burgess

Last modified: Thu Apr 27 12:37:26 MET DST 2000

Linux 2.0.32 will include the IP frag patch for this exploit.  Microsoft has a patch that will correct this problem available at :

ftp://ftp.microsoft.com/bussys/winnt/winnt-public/fixes/usa/nt40/hotfixes-postSP3/simptcp-fix

Date:  Thu, 13 Nov 1997 22:06:15 -0800
From:  G P R
Subject:  Linux IP fragment overlap bug

Helu.

I wrote this post a while back when the bug was first discovered.  It seems as though this bug (and patch) has gotten out, so here it is, in it's entirety.

As it happens, Linux has a serious bug in it's IP fragmentation module.  More specifically, in the fragmentation reassembly code.  More specifically, the bug manifests itself in the `ip_glue()` function....

When Linux reassembles IP fragments to form the original IP datagram, it runs in a loop, copying the payload from all the queued fragments into a newly allocated buffer (which would then normally be passed to the IP layer proper).  >From ip_fragment.c@376:

```
        fp = qp->fragments;
        while(fp != NULL)
        {
                if(count+fp->len > skb->len)
                {
                    error_to_big;
                }
                memcpy((ptr + fp->offset), fp->ptr, fp->len);
                count += fp->len;
                fp = fp->next;
        }
```

While it does check to see if the fragment length is too large, which would have the kernel copy too much data, it doesn't check to see if the fragment length is too small, which would have the kernel copy WAY too data (such is the case if fp->len is < 0).

To see when this happens, we need to look at how Linux adds IP datagrams to the reassembly queue.  From ip_fragment.c@502:

```
        /*
         *      Determine the position of this fragment.
         */

        end = offset + ntohs(iph->tot_len) - ihl;
```

Ok.  That's nice.  Now we have to look at what happens when we have overlaping fragments...  From ip_fragment.c@531:

```
        /*
         *      We found where to put this one.
         *      Check for overlap with preceding fragment, and, if needed,
         *      align things so that any overlaps are eliminated.
         */
```

```
        if (prev != NULL && offset < prev->end)
        {
                i = prev->end - offset;
                offset += i;    /* ptr into datagram */
                ptr += i;       /* ptr into fragment data */
        }
```

    If we find that the current fragment's offset is inside the end of a previous fragment (overlap), we need to (try) align it correctly.  Well, this is fine and good, unless the payload of the current fragment happens to NOT contain enough data to cover the realigning.  In that case, `offset` will end up being larger then `end`.  These two values are passed to `ip_frag_create()` where the length of the fragment data is computed.  From ip_fragment.c@97:

```
        /* Fill in the structure. */
        fp->offset = offset;
        fp->end = end;
        fp->len = end - offset;
```

    This results in fp->len being negative and the memcpy() at the top will end up trying to copy entirely too much data, resulting in a reboot or a halt, depending on how much physical memory you've got.

    We can trigger this normally unlikely event by simply sending 2 specially fragmented IP datagrams.  The first is the 0 offset fragment with a payload of size N, with the MF bit on (data content is irrelevant).  The second is the last fragment (MF == 0) with a positive offset < N and with a payload of < N.

    Every linux implementation I have been able to look at seems to have this problem (1.x - 2.x, including the development kernels).

    Oh, by the way, NT/95 appear to have the bug also.  Try sending 10 - 15 of these fragment combos to an NT/95 machine.

    Special thanks to klepto for bringing the problem to my attention and writing the initial exploit.

            route|daemon9          route@infonexus.com

------[Begin] -- Guby Linux ------------------------------------------------

```
/*
 *  Copyright (c) 1997 route|daemon9   11.3.97
 *
 *  Linux/NT/95 Overlap frag bug exploit
 *
 *  Exploits the overlapping IP fragment bug present in all Linux kernels and
 *  NT 4.0 / Windows 95 (others?)
 *
 *  Based off of:   flip.c by klepto
 *  Compiles on:    Linux, *BSD*
 *
 *  gcc -O2 teardrop.c -o teardrop
 *      OR
 *  gcc -O2 teardrop.c -o teardrop -DSTRANGE_BSD_BYTE_ORDERING_THING
 */

#include
#include
#include
```

```
#include
#include
#include
#include
#include
#include
#include
#include

[snip...]

    fprintf(stderr, "teardrop   route|daemon9\n\n");

[snip...]

    fprintf(stderr, "Death on flaxen wings:\n");
    addr.s_addr = src_ip;
    fprintf(stderr, "From: %15s.%5d\n", inet_ntoa(addr), src_prt);
    addr.s_addr = dst_ip;
    fprintf(stderr, "  To: %15s.%5d\n", inet_ntoa(addr), dst_prt);
    fprintf(stderr, " Amt: %5d\n", count);
    fprintf(stderr, "[ ");

[snip...]

------[End] -- Guby Linux ------------------------------------------------------

    And the patch:

------[Begin] -- Helu Linux ----------------------------------------------------

--- ip_fragment.c       Mon Nov 10 14:58:38 1997
+++ ip_fragment.c.patched       Mon Nov 10 19:18:52 1997
@@ -12,6 +12,7 @@
  *             Alan Cox        :       Split from ip.c , see ip_input.c for history.
  *             Alan Cox        :       Handling oversized frames
  *             Uriel Maimon    :       Accounting errors in two fringe cases.
+ *             route           :       IP fragment overlap bug
  */

 #include
@@ -578,6 +579,22 @@
                         frag_kfree_s(tmp, sizeof(struct ipfrag));
                 }
         }
+
+        /*
+         * Uh-oh.  Some one's playing some park shenanigans on us.
+         * IP fragoverlap-linux-go-b00m bug.
+         * route 11.3.97
+         */
+
+        if (offset > end)
+        {
+                skb->sk = NULL;
+                printk("IP: Invalid IP fragment (offset > end) found from %s\n",
in_ntoa(iph->saddr));
+                kfree_skb(skb, FREE_READ);
+                ip_statistics.IpReasmFails++;
+                ip_free(qp);
```

```
+                return NULL;
+        }

        /*
         *      Insert this fragment in the chain of fragments.

------[End] -- Helu Linux ----------------------------------------------------

EOF

--
        Corporate
                Persuasion
                        Through
                              Internet
                                     Terrorism.
```

FREE DOWNLOADS

- e-TEST suite
- Bean-test
- FirstACT
- OneSight
- CallFlow
- CallMaster
- H.323 CallGen

# Voice/Network Test

## Data Network Test

DATASHEET

**ANVL™**
**Automated Network Validation Library**

Automated Network Validation Library (ANVL) is a software testing tool that validates the protocol implementations and operational robustness of networking devices including layer 3 switches, high-density RA servers, backbone routers and even end nodes. Running on a user's workstation, ANVL quickly and accurately determines how well a network device's protocol implementation adheres to the specifications for the protocol.

Because of its versatility, ANVL has become an essential tool for both product developers and quality assurance engineers. Vendors like Cisco, 3Com, Nortel and Lucent depend on ANVL to thoroughly test more than 20 different protocols on over hundreds of different devices.

Over 100 networking vendors worldwide use ANVL to ensure the protocols in their products are implemented correctly. ANVL is the industry standard for protocol interoperability testing, providing functional, negative, and regression stress testing in one automated software package.

**ANVL saves time.**

In product development and testing, ANVL allows vendors to shorten their release time and get their products into the marketplace sooner. ANVL can be used to streamline the development of new protocol software by providing fast and highly reliable unit testing. It also lets quality assurance staff run tests at any time, even overnight, saving valuable time in the testing of new products.

**ANVL saves money.**

By reducing the demands to continually expand the test network. A single ANVL-equipped workstation can easily take the place of a large, multi-node test network that includes several different types of network devices. The cost of the equivalent test network could easily exceed a quarter of a million dollars.

**ANVL simplifies testing.**

ANVL provides comprehensive pre-written test suites for today's most widely used protocols. Typical of the test suites available are those for the PPP protocol family, Border Gateway Protocol (BGP4), and Open Shortest Path First Protocol (OSPF). A complete listing of all current ANVL test suites is available at the end of this product description.

**ANVL increases confidence.**

ANVL increases confidence in product quality by enabling extensive and

OVERVIEW
PRODUCTS...
- Telephony Feature Test
- Telephony Performance Test
- VoIP Test
- Protocol Conformance Test
- PacketSphere

RESOURCES
NEWS+INFO
SUPPORT/TRAINING
PARTNERS
SUCCESS STORIES
E-MAIL SALES

thorough testing to be performed automatically and without supervision. The result is higher levels of product quality than can be achieved with manual testing.

With ANVL's test results, you can:

- Determine exactly where a device's protocol software does and does not meet the specification.
- Observe how well the device will handle traffic from non-complying network components.
- Determine what effects new development has on existing code through regression testing.

**The ANVL Advantage: Flexible, Intelligent Testing**

ANVL contains unique features not found in other test tools or test processes. These features have been cited by product engineers as invaluable for testing during development, as well as by quality assurance engineers for QA testing.

**ANVL provides flexible, protocol-level testing**

By testing at the protocol level, ANVL is able to control very closely the packets it sends out in order to test a desired situation. For instance, it is able to send out incorrectly formatted packets to test negative situations. It is also able to simulate multiple network nodes, as in stress tests, since it controls the source address of every packet that is sent.

**ANVL sends, receives, and reacts to packets**

ANVL performs its tests as a dialogue: it sends packets to the device being tested, receives the packets sent in response, and analyzes the response to determine the next action to take. This feature allows ANVL to test complicated situations or reactions in a much more intelligent and flexible way than can be done by simple packet generation and capture devices.

**ANVL can test timed events**

Timers implemented in ANVL allow thorough testing of timed events, such as routing updates.p

**ANVL requires minimal equipment**

All a user needs to run ANVL tests is a UNIX or Windows workstation, and a connection to the device to be tested. Ease of set-up makes it possible for a user to be running productive tests soon after ANVL is installed.

**Users can specify the logging level**

By choosing high, medium, or low level output, users can determine how much information to receive about the test as it is being run. High level shows only whether the test passed or failed, medium level gives status information about the test events, and low level displays the actual packets being sent and received.

**ANVL is easy to extend**

With a source license, a user can add new interface types, protocols, and/or tests to their ANVL system.

**Using ANVL Test Suites for Automated Testing**

At ANVL's core is a library of test suites, each based on a specific network protocol. By executing the tests within a given suite, the user is able to determine how well a network protocol implementation conforms to the specification for that protocol. To minimize the time and effort required to use these tests, ANVL includes an automated test system that lets the user set

up and run any number of tests automatically.

A test suite typically includes three types of tests:

- Functional tests that assess conformance to a protocol specification.
- Negative tests that check how a product handles badly formatted packets.
- Stress tests that indicate how well  a device reacts to high traffic situations.

Negative and stress tests play a key role in helping users evaluate how smoothly and robustly a device will behave in real-world network installations. With a comprehensive negative and stress test solution, a network product vendor avoids the risk of shipping products that don't meet performance expectations when installed in customer networks.

Test suites, which are stored in the workstation, are set up and executed automatically by the automated test system, which also resides in the workstation. The user can specify three different levels of test output, ranging from basic pass/fail to comprehensive packet display. Depending on how the protocol and the device under test is connected to the workstation, any of three different test modes may be supported.

**ANVL:  The Key Is Automation**

Automated test processes allow network vendors and purchasers to do more testing with fewer resources. Each test that is written or obtained for an automated test system can be used again and again. This allows companies to do more testing with each release instead of less. With manual testing, as new features are added, existing features are no longer tested, so less testing is done with each release. Automated testing also allows a company to test a product as many times as it wants without fear of burning out its testing staff.

By automating its testing, a company will find it easier to scale up its operations – all it has to do is add more equipment, not more people.  Using ANVL to automate network testing processes reduces equipment needs overall, since ANVL can emulate the appearance of multiple nodes on its test network. ANVL's protocol-level test suites also let a user send packets that are hard or impossible to generate manually, resulting in much more thorough testing.

**ANVL Is Easy to Implement in Any Enviroment**

ANVL's implementation is independent of the type of network being used, so ANVL can run over many different network types. Supported interfaces include Ethernet, serial line, and Sniffer-formatted capture files. In addition, support is currently being added for ATM and Gigabit Ethernet.

ANVL is easy to integrate into existing test processes. Because it has a command-line interface, ANVL can be used in conjunction with text-based automation tools.

ANVL also supports multiple physical interfaces on a single test machine. This makes it possible, for example, to execute tests that exercise two different ports on a router or remote access server.

For more information about using ANVL to automate network test in your environment, please see the ANVL FAQ.

**ANVL Test Suites**

**IP Test Suites**

- IP RIP (v1 and v2) Gateway
- OSPFv2 (RFC1583/2328)
- BGP4 (RFC1771)
- RMON (RFC1757/RFC1513)

**PPP Test Suites**

- Basic PPP (with tests for LCP, PAP, and CHAP)
- IPCP (RFC1332)
- Multilink PPP (RFC1717/RFC1990)
- VJ Test Suite (TC/IP, RFC1144)
- Spanning Tree (IEEE 802.1d)

**Multicasting Test Suites**

- IGMP (RFC2236v2)
- DVMRP (IETF Draft 3)
- PIM (sold as one unit)
- Sparse Mode - IETF Draft#1v2
- Dense Mode - IETF Draft#3v2

**TCP Test Suites**

- Core (RFC 793, 1122)
- Advanced (RFC 2001, 2581, 1191, 2385)
- Performance (RFC 1323, 2018)

**VPN Test Suites**

- PPTP (IETF Draft 2)
- L2TP (RFC2661)
- IPSec AH (RFC2402/2401)
- IPSec ESP (RFC2406)
- IPSec IKE (RFC2409/2408)
- L2TPSec (RFC2661)

More detailed information about ANVL Test Suites is available here.

**Development Toolkits**

The following toolkits including protocols, state machines and several sample tests:

- TCP Development Toolkit
- SNMP Development Toolkit

**Supported Operating Systems/Platforms**

- Sun Solaris 2.5 on Sparc
- Linux RedHat 6.0 on Pentium PC
- Windows NT-4.0 on Pentium PC

**Supported Media/Link Types**

- Ethernet, Gigabit Ethernet, and async serial available on all platforms
- Sync serial available on Linux
- ATM (OC-3) available on Linux and NT

# Notes on Texas Instruments Processors

Prof. Brian L. Evans

---

At present, TI is developing new processors within three digital signal processor families:

- TMS320C2000 (formerly known as TMS320C20)
  - disk drives, e.g. Seagate
- TMS320C5000 (formerly known as TMS320C54)
  - voiceband modems, e.g. modems by 3Com and the modem for the compact Sun-Denshi Online Station for Playstation 2
  - cell phone handsets, e.g. by Nokia and Ericsson
  - portable MP3 players, e.g. Sanyo Internet audio player
  - digital still cameras, e.g. Sony
  - digital video, e.g. JVC's GR-DVM90 e-CyberCam
- TMS320C6000
  - ADSL modems, e.g. TI's ADSL modems
  - cell phone basestations
  - modem banks
  - laser printers

TI has produced many other families of digital signal processors which they still support but for which they are not developing new members of the families. These families include the TMS32010, TMS320C30, TMS320C40, TMS320C50, and TMS320C80. Note that the TMS32010 family does not have a "C" in it because it was originally designed in NMOS and not CMOS.

# Conventional Fixed-Point DSP Processors

The family of conventional fixed-point DSP processors includes the TMS32010, TMS320C20, TMS320C50, TMS320C54, and TMS320C55. These processors have 16-bit data words and 16-bit program words. The 10 (1982) and C20 (1985) fixed-point processors are being widely used in control applications. The C203, a derivative of the C20, was released in 1995 in response to disk drive manufacturers' needs. The C203 delivers 40 MIPS (80 MHz) and costs under $5.00 in volume. The 10 is widely used as essentially a powerful microcontroller. The C24 is dedicated for motion control.

The C54x is a smaller, low-power version of the C50 meant for use in wireless basestations *and* handsets. The C54x instruction set is *not* compatible with the C50. The C54x reminds me of the Motorola 56000 in that it can perform parallel reads:

- 2 data reads from block 1
- 1 data write to block 2

- 1 instruction fetch from block 3

The C54x has a special instruction for Viterbi decoding. Other features include three idle modes (controlled by host processor) to preserve power consumption and flash memory (must write in 2 kword blocks). A C compiler exists. A low-cost C54x DSP Starter Kit (DSK) also exists. The C54x is also used for servo-control in high-end disk drives.

A variation of the C54x, the C54xx family, has 8 Mwords of addressable memory due to the addition of a page pointer. The TMS320C5416 has 128K words of on-chip SRAM and runs at 160 MHz. Applications include Voice over Internet Protocol (VoIP), communications servers, PBX add-ons and other computer telephony and customer premise equipment.

The C55 is in the C5000 family but has lower power consumption than the C54. The TMS320C5509 DSP is targeted for portable handheld Internet appliances. It has an extensive set of on-board peripherals.

- Clock rate: 144/200 MHz (up to 288/400 MIPS)
- On-chip Memory: 128 kw RAM and 32 kw ROM
- Interfaces: USB 1.1 port, I2C, Memory Stick, MMC, SD, three serial ports
- Data converter: on-chip 10-bit ADC

The TMS320C5502 is a low-cost member of the C5000 family for personal systems at $9.95/unit in quantities of 10,000 units:

- Clock rate: * 200 MHz (up to 400 MIPS)
- On-chip Memory: 32 kw DARAM and 16 kw ROM
- Interfaces: UART, I2C, three serial ports

# Conventional Floating-Point DSP Processors

The first two TI floating-point DSP processors were the TMS320C30 (1988) and TMS320C40 processors. These two processors are very similar. The key difference is that the C40 has extra communications features that allows it to be more easily used in parallel. The C44 is a scaled down version of the C40.

The C30 is the base processor. A DSP Starter Kit (DSK) board with the C31 (August, 1996) sells for $99. This is much cheaper than the $750 for the C30 evaluation module (EVM). Like the EVM, the DSK does not come with a compiler. However, an extension to the GNU C compiler generates code for the C30.

The TMS320C32 sells for $10 each with a volume purchase being required. The C32 is used in the Concur Systems Inc. thin Internet data acquisition systems. The TMS320VC33 sells for $5. The 'C33 provides a full 1-Mbits of random access memory (RAM) and delivers 120 MFLOPS. A 150-MHz

version of the 'C33 is also available for $8.

No more C40 derivatives will be developed. The C40 was intended for use in parallel processing. The fixed-point C80 family briefly superseded the C40 for parallel processing, but no more C80 derivatives will be developed. The C80 is described next. The primary TI processor family for parallel processing is the C6x.

# Unconventional DSP Processors

The members of this family include the TMS320C80, TMS320C62x, and TMS320C67x. The C80 contains four fixed-point DSPs plus a RISC on a single chip and is meant for video processing. The reality is that the C80 is too expensive, consumes too much power, and development tools for it are poor. TI is no longer developing new members of the C8x family, but third-party C8x boards and tools are still being developed, e.g. the Genesis board by Matrox.

The C6x (C6000) family is a Very Long Instruction Word (VLIW) Reduced Instruction Set Computer (RISC) Digital Signal Processor (DSP) with eight parallel function units: 6 are ALUs and 2 are multipliers. The C6x has three key members: C6200 and C6400 for 16-bit fixed-point and C6700 for 32-bit floating-point processing. A 32-bit floating-point multiplication takes 4 cycles. The market share for the C6x family hit $1.5 billion as of October 29, 1999.

When TI reports MIPS for the C6000, they are computing RISC MIPS using 8 times the clock rate. These MIPS are *not* DSP processor MIPS. Another useful figure of merit is million multiply-accumulates per second (MMACS), which is 2 x clock rate for the C6200 and C6400.

## C62x Processor

The C62x has 8 arithmetic units (2 multipliers and 6 adders/shifters). Applications include wireless basestations, modem pools, cable modems, remove access servers, digital subscriber loop modems, and wireless PDAs. Members of the family include:

- TMS320C6211: 150 MHz (1200 RISC MIPS) for $25 (in 25K unit quantities); 64 kbits on-chip memory (32 kbits data; 32 kbits program) plus L2 cache (512 kbits)
- TMS320C6201: 167 MHz (1333 RISC MIPS) and 200 MHz (1600 RISC MIPS); 1 Mbit on-chip memory (512 kbits data; 512 kbits program); low-power version C6201B at 200 MHz consumes 1.94 W of power
- TMS320C6202: 250 MHz (2000 RISC MIPS)
- TMS320C6203: 250 MHz (2000 RISC MIPS) and 300 MHz (2400 RISC MIPS); 7 Mbits on-chip memory (3 Mb program; 4 Mb data); used in digital communication systems, including basestations for third-generation wireless communication systems (wireless data networks) and modem banks (a bank of 24 V.90 modems for a T-1 line on a single chip)

For more details, see http://www.ti.com/sc/c62xdsps.

### C67x Processor

It is pin compatible with the 'C62x. The C67x is in volume production. At 100-MHz, the 'C6711 delivers 600 MFLOPS for only $20. A 150-MHz version of the device, also new, increases performance to 900 MFLOPS. The 'C67x family offers a code-compatible roadmap to 3 GFLOPS and beyond. Applications include beamforming base stations, 3-D virtual reality, graphics, speech recognition, radar/sonar, precision instrumentation, and medical imaging.

# Problems with TI Tools

- No code translators between C5x and C20x and between C54x and C6x exist
- No simulators and debuggers are publicly available, except for the C31.
- C compilers are very poor for the traditional fixed-point DSP processors (C2x/C5x/C54x), but relatively poor for the C6000 processors, when compared to C compilers for desktop computers.

---

Last updated 01/07/02. Send comments to  bevans@ece.utexas.edu

RFC: 760
IEN: 128

# DOD STANDARD

# INTERNET PROTOCOL

January 1980

prepared for

Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

by

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

TABLE OF CONTENTS

# PREFACE

This document specifies the DoD Standard Internet Protocol. This document is based on five earlier editions of the ARPA Internet Protocol Specification, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition revises the details security, compartmentation, and precedence features of the internet protocol.

Jon Postel

Editor

January 1980
RFC: 760
IEN: 128
Replaces: IENs 123, 111,
80, 54, 44, 41, 28, 26

DOD STANDARD

# INTERNET PROTOCOL

## 1. INTRODUCTION

### 1.1. Motivation

The Internet Protocol is designed for use in interconnected systems of packet-switched computer communication networks. Such a system has been called a "catenet" [1]. The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. The internet protocol also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks.

### 1.2. Scope

The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There are no mechanisms to promote data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols.

### 1.3. Interfaces

This protocol is called on by host-to-host protocols in an internet environment. This protocol calls on local network protocols to carry the internet datagram to the next gateway or destination host.

For example, a TCP module would call on the internet module to take a TCP segment (including the TCP header and user data) as the data portion of an internet datagram. The TCP module would provide the addresses and other parameters in the internet header to the internet module as arguments of the call. The internet module would then create an internet datagram and call on the local network interface to transmit the internet datagram.

In the ARPANET case, for example, the internet module would call on a local net module which would add the 1822 leader [2] to the internet datagram creating an ARPANET message to transmit to the IMP. The ARPANET address would be derived from the internet address by the local network interface and would be the address of some host in the ARPANET, that host might be a gateway to other networks.

## 1.4. Operation

The internet protocol implements two basic functions: addressing and fragmentation.

The internet modules use the addresses carried in the internet header to transmit internet datagrams toward their destinations. The selection of a path for transmission is called routing.

The internet modules use fields in the internet header to fragment and reassemble internet datagrams when necessary for transmission through "small packet" networks.

The model of operation is that an internet module resides in each host engaged in internet communication and in each gateway that interconnects networks. These modules share common rules for interpreting address fields and for fragmenting and assembling internet datagrams. In addition, these modules (especially in gateways) may have procedures for making routing decisions and other functions.

The internet protocol treats each internet datagram as an independent entity unrelated to any other internet datagram. There are no connections or logical circuits (virtual or otherwise).

The internet protocol uses four key mechanisms in providing its service: Type of Service, Time to Live, Options, and Header Checksum.

The Type of Service is used to indicate the quality of the service desired; this may be thought of as selecting among Interactive, Bulk, or Real Time, for example. The type of service is an abstract or generalized set of parameters which characterize the service choices provided in the networks that make up the internet. This type of service indication is to be used by gateways to select the actual transmission parameters for a particular network, the network to be used for the next hop, or the next gateway when routing an internet datagram.

The Time to Live is an indication of the lifetime of an internet datagram. It is set by the sender of the datagram and reduced at the points along the route where it is processed. If the time to live reaches zero before the internet datagram reaches its destination, the internet datagram is destroyed. The time to live can be thought of as a self destruct time limit.

The Options provide for control functions needed or useful in some situations but unnecessary for the most common communications. The

options include provisions for timestamps, error reports, and special routing.

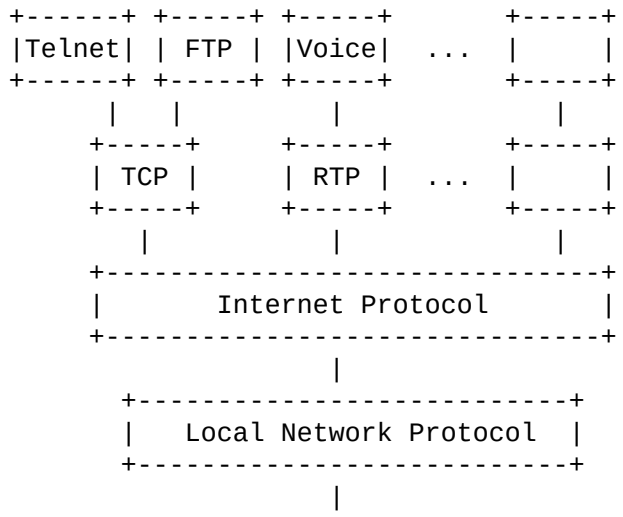
The Header Checksum provides a verification that the information used in processing internet datagram has been transmitted correctly. The data may contain errors. If the header checksum fails, the internet datagram is discarded at once by the entity which detects the error.

The internet protocol does not provide a reliable communication facility. There are no acknowledgments either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is no flow control.

## 2. OVERVIEW

### 2.1. Relation to Other Protocols

The following diagram illustrates the place of the internet protocol in the protocol hierarchy:




Protocol Relationships

Figure 1.

Internet protocol interfaces on one side to the higher level host-to-host protocols and on the other side to the local network protocol.

### 2.2. Model of Operation

The model of operation for transmitting a datagram from one application program to another is illustrated by the following scenario:

We suppose that this transmission will involve one intermediate gateway.

The sending application program prepares its data and calls on its local internet module to send that data as a datagram and passes the destination address and other parameters as arguments of the call.

The internet module prepares a datagram header and attaches the data

to it. The internet module determines a local network address for this internet address, in this case it is the address of a gateway. It sends this datagram and the local network address to the local network interface.

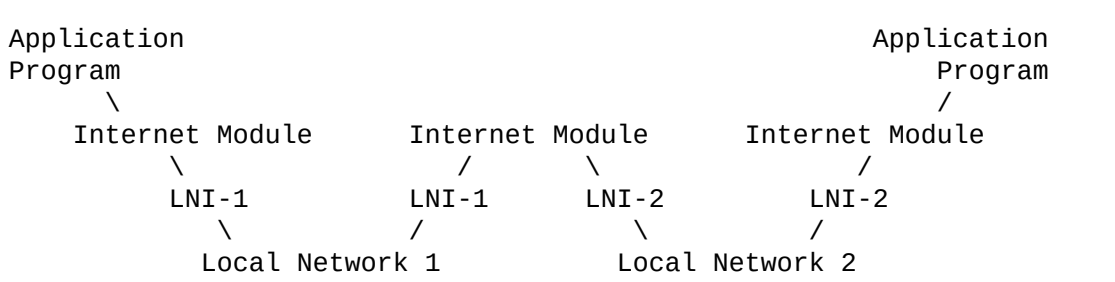
The local network interface creates a local network header, and attaches the datagram to it, then sends the result via the local network.

The datagram arrives at a gateway host wrapped in the local network header, the local network interface strips off this header, and turns the datagram over to the internet module. The internet module determines from the internet address that the datagram should be forwarded to another host in a second network. The internet module determines a local net address for the destination host. It calls on the local network interface for that network to send the datagram.

This local network interface creates a local network header and attaches the datagram sending the result to the destination host.

At this destination host the datagram is stripped of the local net header by the local network interface and handed to the internet module.

The internet module determines that the datagram is for an application program in this host. It passes the data to the application program in response to a system call, passing the source address and other parameters as results of the call.




Transmission Path

Figure 2

## 2.3. Function Description

The function or purpose of Internet Protocol is to move datagrams through an interconnected set of networks. This is done by passing the datagrams from one internet module to another until the destination is reached. The internet modules reside in hosts and gateways in the internet system. The datagrams are routed from one internet module to another through individual networks based on the interpretation of an internet address. Thus, one important mechanism of the internet protocol is the internet address.

In the routing of messages from one internet module to another, datagrams may need to traverse a network whose maximum packet size is smaller than the size of the datagram. To overcome this difficulty, a fragmentation mechanism is provided in the internet protocol.

### Addressing

A distinction is made between names, addresses, and routes [3]. A name indicates what we seek. An address indicates where it is. A route indicates how to get there. The internet protocol deals primarily with addresses. It is the task of higher level (i.e., host-to-host or application) protocols to make the mapping from names to addresses. The internet module maps internet addresses to local net addresses. It is the task of lower level (i.e., local net or gateways) procedures to make the mapping from local net addresses to routes.

Addresses are fixed length of four octets (32 bits). An address begins with a one octet network number, followed by a three octet local address. This three octet field is called the "rest" field.

Care must be taken in mapping internet addresses to local net addresses; a single physical host must be able to act as if it were several distinct hosts to the extent of using several distinct internet addresses. A host should also be able to have several physical interfaces (multi-homing).

That is, a host should be allowed several physical interfaces to the network with each having several logical internet addresses.

Examples of address mappings may be found in reference [4].

### Fragmentation

Fragmentation of an internet datagram may be necessary when it originates in a local net that allows a large packet size and must

traverse a local net that limits packets to a smaller size to reach its destination.

An internet datagram can be marked "don't fragment." Any internet datagram so marked is not to be internet fragmented under any circumstances. If internet datagram marked don't fragment cannot be delivered to its destination without fragmenting it, it is to be discarded instead.

Fragmentation, transmission and reassembly across a local network which is invisible to the internet protocol module is called intranet fragmentation and may be used [5].

The internet fragmentation and reassembly procedure needs to be able to break a datagram into an almost arbitrary number of pieces that can be later reassembled. The receiver of the fragments uses the identification field to ensure that fragments of different datagrams are not mixed. The fragment offset field tells the receiver the position of a fragment in the original datagram. The fragment offset and length determine the portion of the original datagram covered by this fragment. The more-fragments flag indicates (by being reset) the last fragment. These fields provide sufficient information to reassemble datagrams.

The identification field is used to distinguish the fragments of one datagram from those of another. The originating protocol module of an internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system. The originating protocol module of a complete datagram sets the more-fragments flag to zero and the fragment offset to zero.

To fragment a long internet datagram, an internet protocol module (for example, in a gateway), creates two new internet datagrams and copies the contents of the internet header fields from the long datagram into both new internet headers. The data of the long datagram is divided into two portions on a 8 octet (64 bit) boundary (the second portion might not be an integral multiple of 8 octets, but the first must be). Call the number of 8 octet blocks in the first portion NFB (for Number of Fragment Blocks). The first portion of the data is placed in the first new internet datagram, and the total length field is set to the length of the first datagram. The more-fragments flag is set to one. The second portion of the data is placed in the second new internet datagram, and the total length field is set to the length of the second datagram. The more-fragments flag carries the same value as the long datagram. The fragment offset field of the second new internet

datagram is set to the value of that field in the long datagram plus NFB.

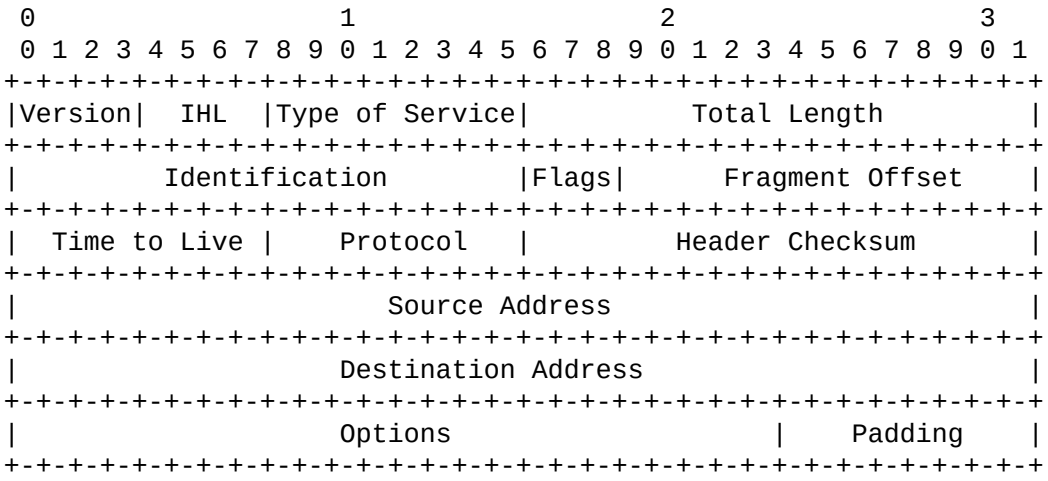
This procedure can be generalized for an n-way split, rather than the two-way split described.

To assemble the fragments of an internet datagram, an internet protocol module (for example at a destination host) combines internet datagram that all have the same value for the four fields: identification, source, destination, and protocol. The combination is done by placing the data portion of each fragment in the relative position indicated by the fragment offset in that fragment's internet header. The first fragment will have the fragment offset zero, and the last fragment will have the more-fragments flag reset to zero.

## 3. SPECIFICATION

### 3.1. Internet Header Format

A summary of the contents of the internet header follows:




Example Internet Datagram Header

Figure 3.

Note that each tick mark represents one bit position.

Version: 4 bits

The Version field indicates the format of the internet header. This document describes version 4.

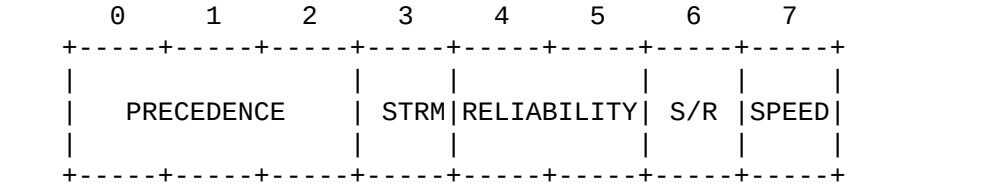
IHL: 4 bits

Internet Header Length is the length of the internet header in 32 bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.

Type of Service: 8 bits

The Type of Service provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important than other traffic. A few networks offer a Stream service, whereby one can achieve a smoother service at some cost. Typically this involves the reservation of resources within the network. Another choice involves a low-delay vs. high-reliability trade off. Typically networks invoke more complex (and delay producing) mechanisms as the need for reliability increases.

Bits 0-2: Precedence.
Bit 3: Stream or Datagram.
Bits 4-5: Reliability.
Bit 6: Speed over Reliability.
Bits 7: Speed.




| PRECEDENCE | STRM | RELIABILITY | S/R | SPEED |
|---|---|---|---|---|
| 111-Flash Override | 1-STREAM | 11-highest | 1-speed | 1-high |
| 110-Flash | 0-DTGRM | 10-higher | 0-rlblt | 0-low |
| 11X-Immediate | | 01-lower | | |
| 01X-Priority | | 00-lowest | | |
| 00X-Routine | | | | |

The type of service is used to specify the treatment of the datagram during its transmission through the internet system. In the discussion (section 3.2) below, a chart shows the relationship of the internet type of service to the actual service provided on the ARPANET, the SATNET, and the PRNET.

Total Length: 16 bits

Total Length is the length of the datagram, measured in octets, including internet header and data. This field allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole

or in fragments). It is recommended that hosts only send datagrams larger than 576 octets if they have assurance that the destination is prepared to accept the larger datagrams.

The number 576 is selected to allow a reasonable sized data block to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximal internet header is 60 octets, and a typical internet header is 20 octets, allowing a margin for headers of higher level protocols.

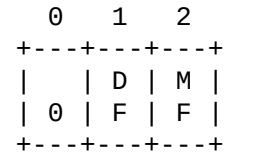
Identification: 16 bits

An identifying value assigned by the sender to aid in assembling the fragments of a datagram.

Flags: 3 bits

Various Control Flags.

Bit 0: reserved, must be zero
Bit 1: Don't Fragment This Datagram (DF).
Bit 2: More Fragments Flag (MF).




Fragment Offset: 13 bits

This field indicates where in the datagram this fragment belongs. The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

Time to Live: 8 bits

This field indicates the maximum time the datagram is allowed to remain the internet system. If this field contains the value zero, then the datagram should be destroyed. This field is modified in internet header processing. The time is measured in units of seconds. The intention is to cause undeliverable datagrams to be discarded.

Protocol: 8 bits

This field indicates the next level protocol used in the data portion of the internet datagram. The values for various protocols are specified in reference [6].

Header Checksum: 16 bits

A checksum on the header only. Since some header fields may change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed.

The checksum algorithm is:

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

Source Address: 32 bits

The source address. The first octet is the Source Network, and the following three octets are the Source Local Address.

Destination Address: 32 bits

The destination address. The first octet is the Destination Network, and the following three octets are the Destination Local Address.

Options: variable

The option field is variable in length. There may be zero or more options. There are two cases for the format of an option:

Case 1: A single octet of option-type.

Case 2: An option-type octet, an option-length octet, and the actual option-data octets.

The option-length octet counts the option-type octet and the option-length octet as well as the option-data octets.

The option-type octet is viewed as having 3 fields:

1 bit reserved, must be zero
2 bits option class,
5 bits option number.

The option classes are:

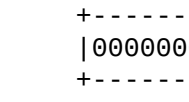
0 = control
1 = internet error
2 = experimental debugging and measurement
3 = reserved for future use

The following internet options are defined:

| CLASS | NUMBER | LENGTH | DESCRIPTION |
|---|---|---|---|
| 0 | 0 | - | End of Option list. This option occupies only 1 octet; it has no length octet. |
| 0 | 1 | - | No Operation. This option occupies only 1 octet; it has no length octet. |
| 0 | 2 | 4 | Security. Used to carry Security, and user group (TCC) information compatible with DOD requirements. |
| 0 | 3 | var. | Source Routing. Used to route the internet datagram based on information supplied by the source. |
| 0 | 7 | var. | Return Route. Used to record the route an internet datagram takes. |
| 0 | 8 | 4 | Stream ID. Used to carry the stream identifier. |
| 1 | 1 | var. | General Error Report. Used to report errors in internet datagram processing. |
| 2 | 4 | 6 | Internet Timestamp. |
| 2 | 5 | 6 | Satellite Timestamp. |

Specific Option Definitions

End of Option List




Type=0

This option indicates the end of the option list. This might not coincide with the end of the internet header according to the internet header length. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the internet header.

May be copied, introduced, or deleted on fragmentation.

No Operation

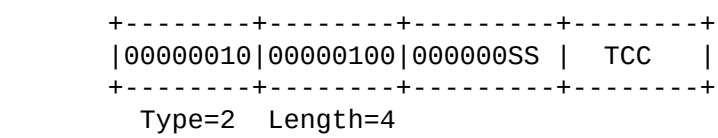



This option may be used between options, for example, to align the beginning of a subsequent option on a 32 bit boundary.

May be copied, introduced, or deleted on fragmentation.

Security

This option provides a way for DOD hosts to send security and TCC (closed user groups) parameters through networks whose transport leader does not contain fields for this information. The format for this option is as follows:




Security: 2 bits

Specifies one of 4 levels of security

11-top secret
10-secret
01-confidential
00-unclassified

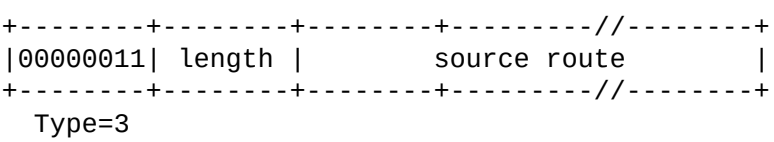
Transmission Control Code: 8 bits

Provides a means to compartmentalize traffic and define controlled communities of interest among subscribers.

Note that this option does not require processing by the internet module but does require that this information be passed to higher level protocol modules. The security and TCC information might be used to supply class level and compartment information for transmitting datagrams into or through AUTODIN II.

Must be copied on fragmentation.

Source Route




The source route option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination.

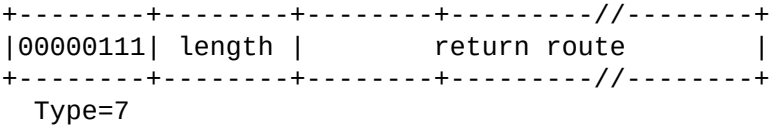
The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, as well as length-2 octets of source route data.

A source route is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. The length defaults to two, which indicates the source route is empty and the remaining routing is to be based on the destination address field.

If the address in destination address field has been reached and this option's length is not two, the next address in the source route replaces the address in the destination address field, and is deleted from the source route and this option's length is reduced by four. (The Internet Header Length Field must be changed also.)

Must be copied on fragmentation.

Return Route




The return route option provides a means to record the route of an internet datagram.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, as well as length-2 octets of return route data.

A return route is composed of a series of internet addresses. The length defaults to two, which indicates the return route is empty.

When an internet module routes a datagram it checks to see if the return route option is present. If it is, it inserts its own internet address as known in the environment into which this datagram is being forwarded into the return route at the front of the address string and increments the length by four.

Not copied on fragmentation, goes in first fragment only.

Stream Identifier

```
+--------+--------+---------+--------+
|00001000|00000010|     Stream ID    |
+--------+--------+---------+--------+
  Type=8  Length=4
```

This option provides a way for the 16-bit SATNET stream identifier to be carried through networks that do not support the stream concept.

Must be copied on fragmentation.

General Error Report

```
+--------+--------+--------+--------+--------+----//----+
|00100001| length |err code|        id       |          |
+--------+--------+--------+--------+--------+----//----+
 Type=33
```

The general error report is used to report an error detected in processing an internet datagram to the source internet module of that datagram. The "err code" indicates the type of error detected, and the "id" is copied from the identification field of the datagram in error, additional octets of error information may be present depending on the err code.

If an internet datagram containing the general error report option is found to be in error or must be discarded, no error report is sent.

ERR CODE:

0 - Undetermined Error, used when no information is available about the type of error or the error does not fit any defined class. Following the id should be as much of the datagram (starting with the internet header) as fits in the option space.

1 - Datagram Discarded, used when specific information is

available about the reason for discarding the datagram can be reported. Following the id should be the original (4-octets) destination address, and the (1-octet) reason.

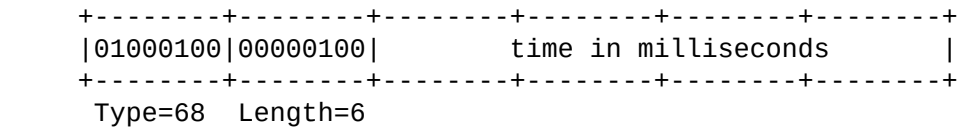
| Reason | Description |
|---|---|
| 0 | No Reason |
| 1 | No One Wants It - No higher level protocol or application program at destination wants this datagram. |
| 2 | Fragmentation Needed & DF - Cannot deliver with out fragmenting and has don't fragment bit set. |
| 3 | Reassembly Problem - Destination could not reassemble due to missing fragments when time to live expired. |
| 4 | Gateway Congestion - Gateway discarded datagram due to congestion. |

The error report is placed in a datagram with the following values in the internet header fields:

Version: Same as the datagram in error.
IHL: As computed.
Type of Service: Zero.
Total Length: As computed.
Identification: A new identification is selected.
Flags: Zero.
Fragment Offset: Zero.
Time to Live: Sixty.
Protocol: Same as the datagram in error.
Header Checksum: As computed.
Source Address: Address of the error reporting module.
Destination Address: Source address of the datagram in error.
Options: The General Error Report Option.
Padding: As needed.

Not copied on fragmentation, goes with first fragment.

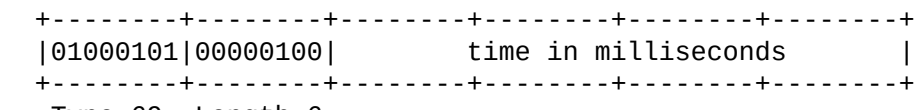
Internet Timestamp




The data of the timestamp is a 32 bit time measured in milliseconds.

Not copied on fragmentation, goes with first fragment

Satellite Timestamp




The data of the timestamp is a 32 bit time measured in milliseconds.

Not copied on fragmentation, goes with first fragment

Padding: variable

The internet header padding is used to ensure that the internet header ends on a 32 bit boundary. The padding is zero.

## 3.2. Discussion

The implementation of a protocol must be robust. Each implementation must expect to interoperate with others created by different individuals. While the goal of this specification is to be explicit about the protocol there is the possibility of differing interpretations. In general, an implementation should be conservative in its sending behavior, and liberal in its receiving behavior. That is, it should be careful to send well-formed datagrams, but should accept any datagram that it can interpret (e.g., not object to technical errors where the meaning is still clear).

The basic internet service is datagram oriented and provides for the fragmentation of datagrams at gateways, with reassembly taking place at the destination internet protocol module in the destination host. Of course, fragmentation and reassembly of datagrams within a network or by private agreement between the gateways of a network is also allowed since this is transparent to the internet protocols and the higher-level protocols. This transparent type of fragmentation and reassembly is termed "network-dependent" (or intranet) fragmentation and is not discussed further here.

Internet addresses distinguish sources and destinations to the host level and provide a protocol field as well. It is assumed that each protocol will provide for whatever multiplexing is necessary within a host.

Addressing

The 8 bit network number, which is the first octet of the address, has a value as specified in reference [6].

The 24 bit local address, assigned by the local network, should allow for a single physical host to act as several distinct internet hosts. That is, there should be mapping between internet host addresses and network/host interfaces that allows several internet addresses to correspond to one interface. It should also be allowed for a host to have several physical interfaces and to treat the datagrams from several of them as if they were all addressed to a single host. Address mappings between internet addresses and addresses for ARPANET, SATNET, PRNET, and other networks are described in reference [4].

Fragmentation and Reassembly.

The internet identification field (ID) is used together with the source and destination address, and the protocol fields, to identify datagram fragments for reassembly.

The More Fragments flag bit (MF) is set if the datagram is not the last fragment. The Fragment Offset field identifies the fragment location, relative to the beginning of the original unfragmented datagram. Fragments are counted in units of 8 octets. The fragmentation strategy is designed so than an unfragmented datagram has all zero fragmentation information (MF = 0, fragment offset = 0). If an internet datagram is fragmented, its data portion must be broken on 8 octet boundaries.

This format allows 2**13 = 8192 fragments of 8 octets each for a total of 65,536 octets. Note that this is consistent with the the datagram total length field.

When fragmentation occurs, some options are copied, but others remain with the first fragment only.

Every internet module must be able to forward a datagram of 68 octets without further fragmentation. This is because an internet header may be up to 60 octets, and the minimum fragment is 8 octets.

Every internet destination must be able to receive a datagram of 576 octets either in one piece or in fragments to be reassembled.

The fields which may be affected by fragmentation include:

(1) options field
(2) more fragments flag
(3) fragment offset
(4) internet header length field
(5) total length field
(6) header checksum

If the Don't Fragment flag (DF) bit is set, then internet fragmentation of this datagram is NOT permitted, although it may be discarded. This can be used to prohibit fragmentation in cases where the receiving host does not have sufficient resources to reassemble internet fragments.

General notation in the following pseudo programs: "=<" means "less than or equal", "#" means "not equal", "=" means "equal", "<-" means "is set to". Also, "x to y" includes x and excludes y; for example, "4 to 7" would include 4, 5, and 6 (but not 7).

Fragmentation Procedure

The maximum sized datagram that can be transmitted through the next network is called the maximum transmission unit (MTU).

If the total length is less than or equal the maximum transmission unit then submit this datagram to the next step in datagram processing; otherwise cut the datagram into two fragments, the first fragment being the maximum size, and the second fragment being the rest of the datagram. The first fragment is submitted to the next step in datagram processing, while the second fragment is submitted to this procedure in case it still too large.

Notation:

FO - Fragment Offset
IHL - Internet Header Length
MF - More Fragments flag
TL - Total Length
OFO - Old Fragment Offset
OIHL - Old Internet Header Length
OMF - Old More Fragments flag
OTL - Old Total Length
NFB - Number of Fragment Blocks
MTU - Maximum Transmission Unit

Procedure:

```
IF TL =< MTU THEN Submit this datagram to the next step
     in datagram processing ELSE
To produce the first fragment:
(1)  Copy the original internet header;
(2)  OIHL <- IHL; OTL <- TL; OFO <- FO; OMF <- MF;
(3)  NFB <- (MTU-IHL*4)/8;
(4)  Attach the first NFB*8 data octets;
(5)  Correct the header:
     MF <- 1;  TL <- (IHL*4)+(NFB*8);
     Recompute Checksum;
(6)  Submit this fragment to the next step in
     datagram processing;
To produce the second fragment:
(7)  Selectively copy the internet header (some options
     are not copied, see option definitions);
(8)  Append the remaining data;
(9)  Correct the header:
     IHL <- (((OIHL*4)-(length of options not copied))+3)/4;
     TL <- OTL - NFB*8 - (OIHL-IHL)*4);
     FO <- OFO + NFB;  MF <- OMF;  Recompute Checksum;
(10) Submit this fragment to the fragmentation test; DONE.
```

Reassembly Procedure

For each datagram the buffer identifier is computed as the concatenation of the source, destination, protocol, and identification fields. If this is a whole datagram (that is both the fragment offset and the more fragments fields are zero), then any reassembly resources associated with this buffer identifier are released and the datagram is forwarded to the next step in datagram processing.

If no other fragment with this buffer identifier is on hand then reassembly resources are allocated. The reassembly resources consist of a data buffer, a header buffer, a fragment block bit table, a total data length field, and a timer. The data from the fragment is placed in the data buffer according to its fragment offset and length, and bits are set in the fragment block bit table corresponding to the fragment blocks received.

If this is the first fragment (that is the fragment offset is zero) this header is placed in the header buffer. If this is the last fragment ( that is the more fragments field is zero) the total data length is computed. If this fragment completes the datagram (tested by checking the bits set in the fragment block table), then the datagram is sent to the next step in datagram

processing; otherwise the timer is set to the maximum of the current timer value and the value of the time to live field from this fragment; and the reassembly routine gives up control.

If the timer runs out, the all reassembly resources for this buffer identifier are released. The initial setting of the timer is a lower bound on the reassembly waiting time. This is because the waiting time will be increased if the Time to Live in the arriving fragment is greater than the current timer value but will not be decreased if it is less. The maximum this timer value could reach is the maximum time to live (approximately 4.25 minutes). The current recommendation for the initial timer setting is 15 seconds. This may be changed as experience with this protocol accumulates. Note that the choice of this parameter value is related to the buffer capacity available and the data rate of the transmission medium; that is, data rate times timer value equals buffer size (e.g., 10Kb/s X 15s = 150Kb).

Notation:

```
  FO    -  Fragment Offset
  IHL   -  Internet Header Length
  MF    -  More Fragments flag
  TTL   -  Time To Live
  NFB   -  Number of Fragment Blocks
  TL    -  Total Length
  TDL   -  Total Data Length
  BUFID -  Buffer Identifier
  RCVBT -  Fragment Received Bit Table
  TLB   -  Timer Lower Bound
```

Procedure:

```
(1)  BUFID <- source|destination|protocol|identification;
(2)  IF FO = 0 AND MF = 0
(3)     THEN IF buffer with BUFID is allocated
(4)             THEN flush all reassembly for this BUFID;
(5)          Submit datagram to next step; DONE.
(6)     ELSE IF no buffer with BUFID is allocated
(7)             THEN allocate reassembly resources
                     with BUFID;
                     TIMER <- TLB; TDL <- 0;
(8)          put data from fragment into data buffer with
             BUFID from octet FO*8 to
                                  octet (TL-(IHL*4))+FO*8;
(9)          set RCVBT bits from FO
                                 to FO+((TL-(IHL*4)+7)/8);
(10)         IF MF = 0 THEN TDL <- TL-(IHL*4)+(FO*8)
(11)         IF FO = 0 THEN put header in header buffer
(12)         IF TDL # 0
(13)          AND all RCVBT bits from 0
                                      to (TDL+7)/8 are set
(14)            THEN TL <- TDL+(IHL*4)
(15)                 Submit datagram to next step;
(16)                 free all reassembly resources
                     for this BUFID; DONE.
(17)         TIMER <- MAX(TIMER,TTL);
(18)         give up until next fragment or timer expires;
(19) timer expires: flush all reassembly with this BUFID; DONE.
```

In the case that two or more fragments contain the same data either identically or through a partial overlap, this procedure will use the more recently arrived copy in the data buffer and datagram delivered.

## Identification

The choice of the Identifier for a datagram is based on the need to provide a way to uniquely identify the fragments of a particular datagram. The protocol module assembling fragments judges fragments to belong to the same datagram if they have the same source, destination, protocol, and Identifier. Thus, the sender must choose the Identifier to be unique for this source, destination pair and protocol for the time the datagram (or any fragment of it) could be alive in the internet.

It seems then that a sending protocol module needs to keep a table of Identifiers, one entry for each destination it has communicated with in the last maximum packet lifetime for the internet.

However, since the Identifier field allows 65,536 different values, some host may be able to simply use unique identifiers independent of destination.

It is appropriate for some higher level protocols to choose the identifier. For example, TCP protocol modules may retransmit an identical TCP segment, and the probability for correct reception would be enhanced if the retransmission carried the same identifier as the original transmission since fragments of either datagram could be used to construct a correct TCP segment.

Type of Service

The type of service (TOS) is for internet service quality selection. The type of service is specified along the abstract parameters precedence, reliability, and speed. A further concern is the possibility of efficient handling of streams of datagrams. These abstract parameters are to be mapped into the actual service parameters of the particular networks the datagram traverses.

Precedence. An independent measure of the importance of this datagram.

Stream or Datagram. Indicates if there will be other datagrams from this source to this destination at regular frequent intervals justifying the maintenance of stream processing information.

Reliability. A measure of the level of effort desired to ensure delivery of this datagram.

Speed over Reliability. Indicates the relative importance of speed and reliability when a conflict arises in meeting the pair of requests.

Speed. A measure of the importance of prompt delivery of this datagram.

For example, the ARPANET has a priority bit, and a choice between "standard" messages (type 0) and "uncontrolled" messages (type 3), (the choice between single packet and multipacket messages can also be considered a service parameter). The uncontrolled messages tend to be less reliably delivered and suffer less delay. Suppose an internet datagram is to be sent through the ARPANET. Let the internet type of service be given as:

```
  Precedence:    5
  Stream:        0
  Reliability:   1
  S/R:           1
  Speed:         1
```

The mapping of these parameters to those available for the ARPANET would be to set the ARPANET priority bit on since the Internet priority is in the upper half of its range, to select uncontrolled messages since the speed and reliability requirements are equal and speed is preferred.

The following chart presents the recommended mappings from the internet protocol type of service into the service parameters actually available on the ARPANET, the PRNET, and the SATNET:

| Application | INTERNET | ARPANET | PRNET | SATNET |
|---|---|---|---|---|
| TELNET on TCP | S/D:stream<br>R:normal<br>S/R:speed<br>S:fast | T: 3<br>S: S | R: ptp<br>A: no | T: block<br>D: min<br>H: inf<br>R: no |
| FTP on TCP | S/D:stream<br>R:normal<br>S/R:rlblt<br>S:normal | T: 0<br>S: M | R: ptp<br>A: no | T: block<br>D: normal<br>H: inf<br>R: no |
| interactive narrow band speech | S/D:strm*<br>R:least<br>P:speed<br>S:asap | T: 3<br>S: S | R: ptp<br>A: no | T: stream<br>D: min<br>H: short<br>R: no |
| datagram | S/D:dtgrm<br>R:normal<br>S/R:speed<br>S:fast | T: 3 or 0<br>S: S or M | R:station<br>A: no | T: block<br>D: min<br>H: short<br>R: no |

```
 key:    S/D=strm/dtgrm   T=type     R=route  T=type
         R=reliability    S=size     A=ack    D=delay
         S/R=speed/rlblt                      H=holding time
         S=speed                              R=reliability
         *=requires stream set up
```

Time to Live

The time to live is set by the sender to the maximum time the datagram is allowed to be in the internet system. If the datagram is in the internet system longer than the time to live, then the datagram should be destroyed. This field should be decreased at each point that the internet header is processed to reflect the time spent processing the datagram. Even if no local information is available on the time actually spent, the field should be decremented by 1. The time is measured in units of seconds (i.e. the value 1 means one second). Thus, the maximum time to live is 255 seconds or 4.25 minutes.

Options

The options are just that, optional. That is, the presence or absence of an option is the choice of the sender, but each internet module must be able to parse every option. There can be several options present in the option field.

The options might not end on a 32-bit boundary. The internet header should be filled out with octets of zeros. The first of these would be interpreted as the end-of-options option, and the remainder as internet header padding.

Every internet module must be able to act on the following options: End of Option List (0), No Operation (1), Source Route (3), Return Route (7), General Error Report (33), and Internet Timestamp (68). The Security Option (2) is required only if classified or compartmented traffic is to be passed.

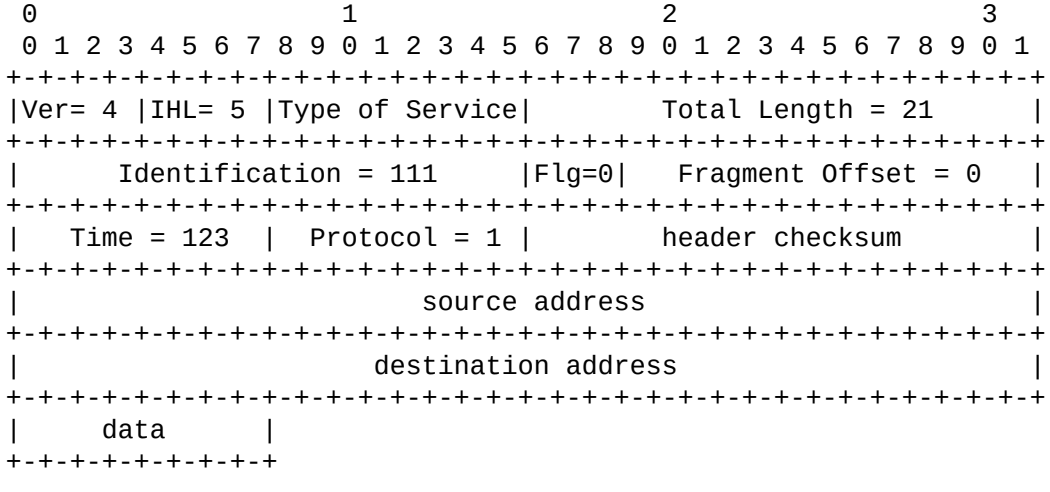
Checksum

The internet header checksum is recomputed if the internet header is changed. For example, a reduction of the time to live, additions or changes to internet options, or due to fragmentation. This checksum at the internet level is intended to protect the internet header fields from transmission errors.

## 3.3. Examples & Scenarios

Example 1:

This is an example of the minimal data carrying internet datagram:




Example Internet Datagram

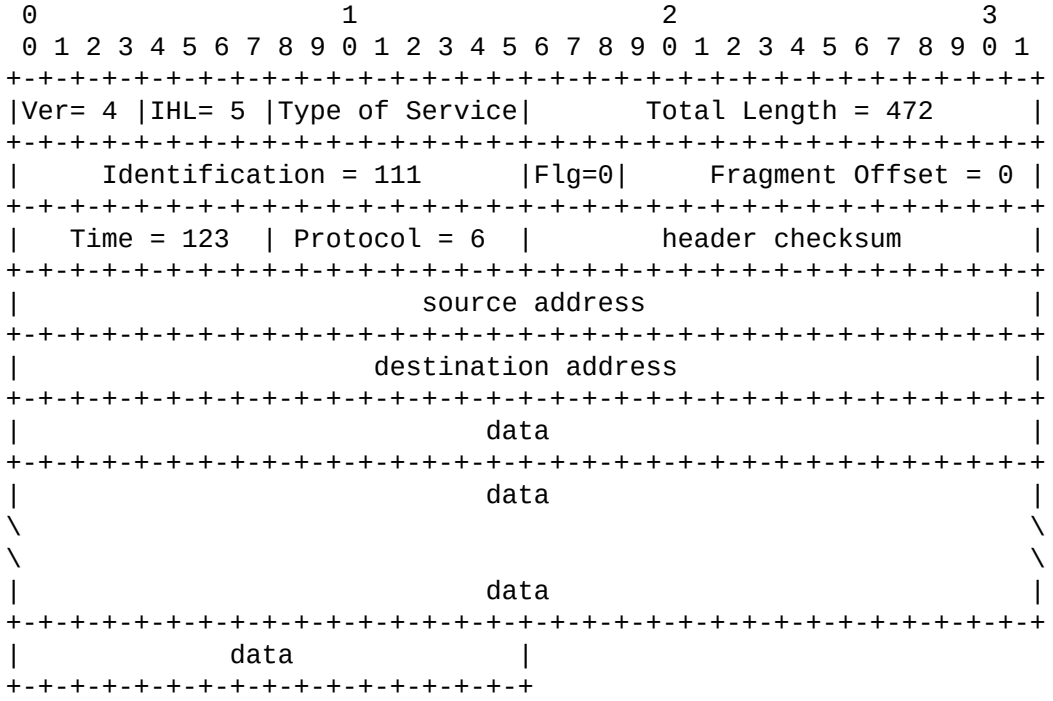
Figure 4.

Note that each tick mark represents one bit position.

This is a internet datagram in version 4 of internet protocol; the internet header consists of five 32 bit words, and the total length of the datagram is 21 octets. This datagram is a complete datagram (not a fragment).

Example 2:

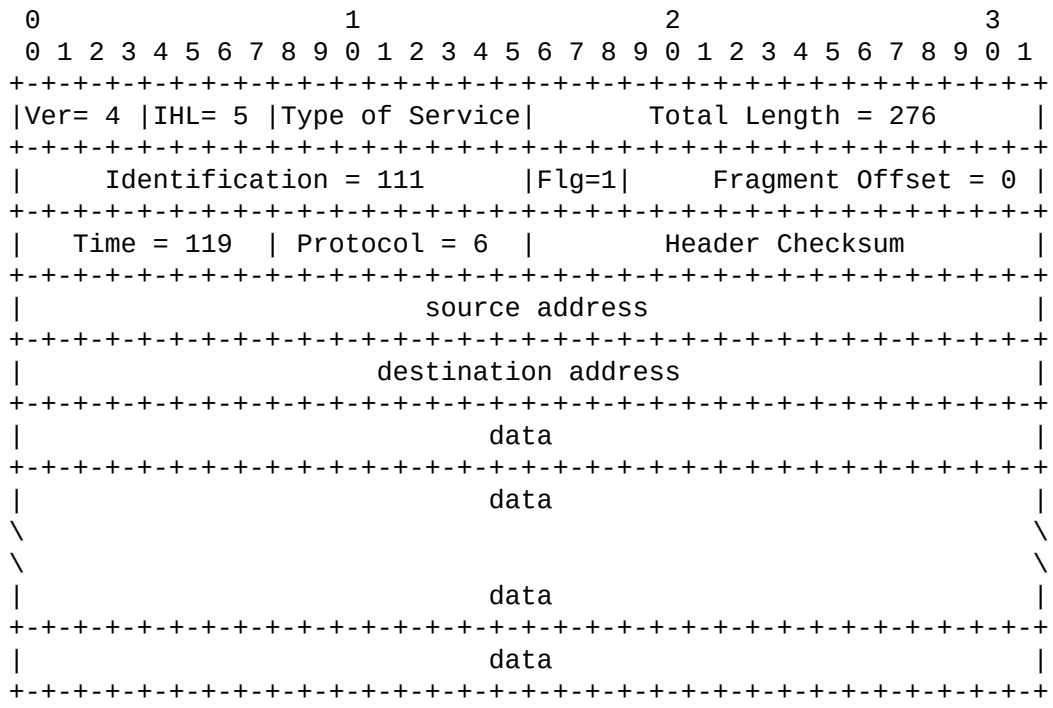
In this example, we show first a moderate size internet datagram (552 data octets), then two internet fragments that might result from the fragmentation of this datagram if the maximum sized transmission allowed were 280 octets.




Example Internet Datagram

Figure 5.

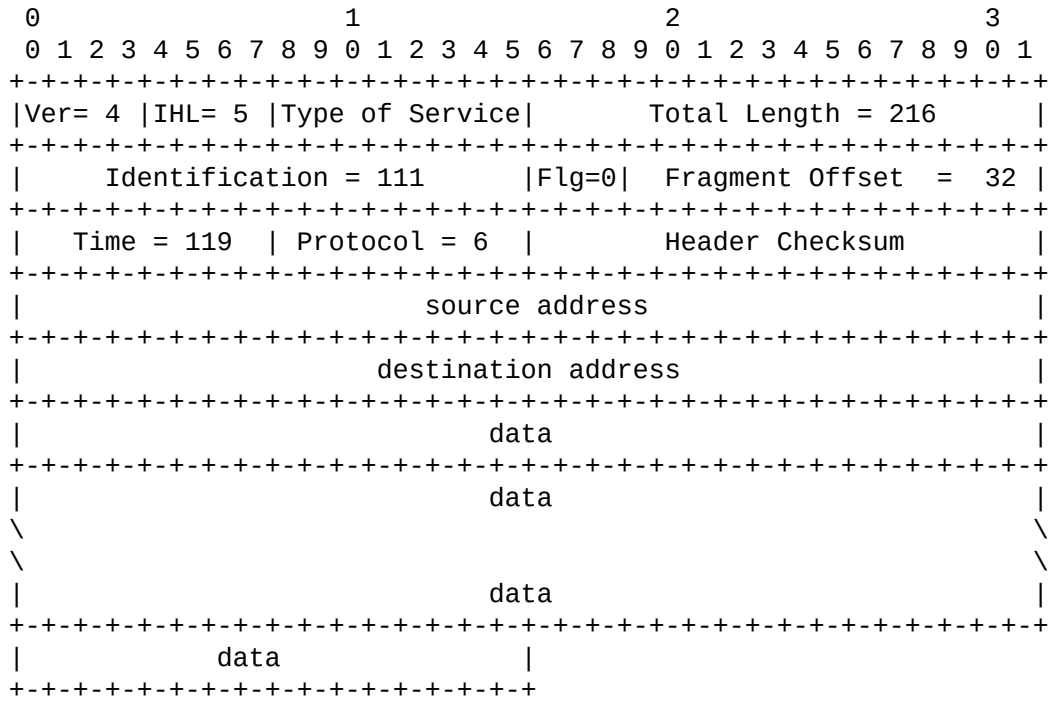
Now the first fragment that results from splitting the datagram after 256 data octets.




Example Internet Fragment

Figure 6.

And the second fragment.

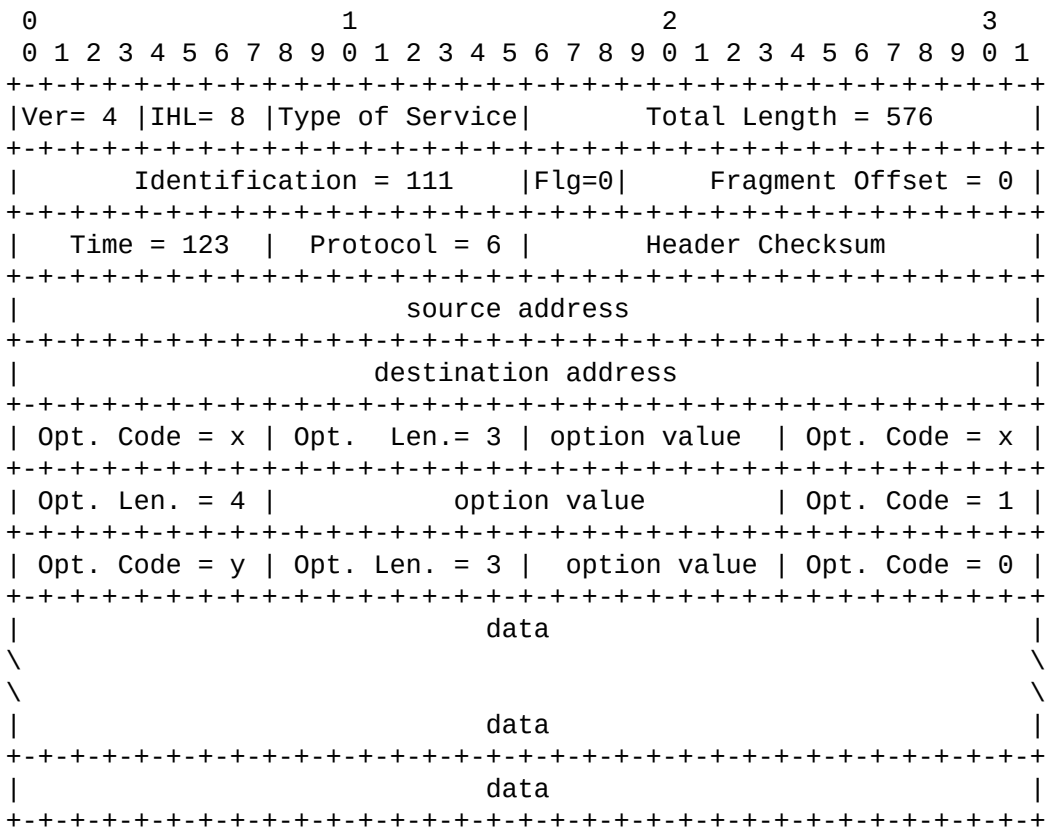



Example Internet Fragment

Figure 7.

Example 3:

Here, we show an example of a datagram containing options:




Example Internet Datagram

Figure 8.

## 3.4. Interfaces

Internet protocol interfaces on one side to the local network and on the other side to either a higher level protocol or an application program. In the following, the higher level protocol or application program (or even a gateway program) will be called the "user" since it is using the internet module. Since internet protocol is a datagram protocol, there is minimal memory or state maintained between datagram transmissions, and each call on the internet protocol module by the user supplies all the necessary information.

For example, the following two calls satisfy the requirements for the user to internet protocol module communication ("=>" means returns):

```
SEND (dest, TOS, TTL, BufPTR, len, Id, DF, options => result)

  where:

    dest = destination address
    TOS = type of service
    TTL = time to live
    BufPTR = buffer pointer
    len = length of buffer
    Id  = Identifier
    DF = Don't Fragment
    options = option data
    result = response
      OK = datagram sent ok
      Error = error in arguments or local network error

RECV (BufPTR => result, source, dest, prot, TOS, len)

  where:

    BufPTR = buffer pointer
    result = response
      OK = datagram received ok
      Error = error in arguments
    source = source address
    dest = destination address
    prot = protocol
    TOS = type of service
    len = length of buffer
```

When the user sends a datagram, it executes the SEND call supplying all the arguments. The internet protocol module, on receiving this call, checks the arguments and prepares and sends the message. If the arguments are good and the datagram is accepted by the local network, the call returns successfully. If either the arguments are bad, or the datagram is not accepted by the local network, the call returns unsuccessfully. On unsuccessful returns, a reasonable report should be made as to the cause of the problem, but the details of such reports are up to individual implementations.

When a datagram arrives at the internet protocol module from the local network, either there is a pending RECV call from the user addressed or there is not. In the first case, the pending call is satisfied by passing the information from the datagram to the user. In the second case, the user addressed is notified of a pending datagram. If the

user addressed does not exist, an error datagram is returned to the sender, and the data is discarded.

The notification of a user may be via a pseudo interrupt or similar mechanism, as appropriate in the particular operating system environment of the implementation.

A user's RECV call may then either be immediately satisfied by a pending datagram, or the call may be pending until a datagram arrives.

An implementation may also allow or require a call to the internet module to indicate interest in or reserve exclusive use of a class of datagrams (e.g., all those with a certain value in the protocol field).

## GLOSSARY

1822
: BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

ARPANET message
: The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet
: A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).

Destination
: The destination address, an internet header field.

DF
: The Don't Fragment bit carried in the flags field.

Flags
: An internet header field carrying various control flags.

Fragment Offset
: This internet header field indicates where in the internet datagram a fragment belongs.

header
: Control information at the beginning of a message, segment, datagram, packet or block of data.

Identification
: An internet header field carrying the identifying value assigned by the sender to aid in assembling the fragments of a datagram.

IHL
: The internet header field Internet Header Length is the length of the internet header measured in 32 bit words.

IMP
: The Interface Message Processor, the packet switch of the ARPANET.

Internet Address
: A four octet (32 bit) source or destination address consisting of a Network field and a Local Address field.

internet fragment
: A portion of the data of an internet datagram with an internet header.

internet datagram
: The unit of data exchanged between a pair of internet modules (includes the internet header).

ARPANET leader
: The control information on an ARPANET message at the host-IMP interface.

Local Address
: The address of a host within a network. The actual mapping of an internet local address on to the host addresses in a network is quite general, allowing for many to one mappings.

MF
: The More-Fragments Flag carried in the internet header flags field.

module
: An implementation, usually in software, of a protocol or other procedure.

more-fragments flag
: A flag indicating whether or not this internet datagram contains the end of an internet datagram, carried in the internet header Flags field.

NFB
: The Number of Fragment Blocks in a the data portion of an internet fragment. That is, the length of a portion of data measured in 8 octet units.

octet
: An eight bit byte.

Options
: The internet header Options field may contain several options, and each option may be several octets in length. The options are used primarily in testing situations, for example to carry timestamps.

Padding
: The internet header Padding field is used to ensure that the data begins on 32 bit word boundary. The padding is zero.

Protocol
: In this document, the next higher level protocol identifier, an internet header field.

Rest
: The 3 octet (24 bit) local address portion of an Internet Address.

RTP
: Real Time Protocol: A host-to-host protocol for communication of time critical information.

Source
: The source address, an internet header field.

TCP
: Transmission Control Protocol: A host-to-host protocol for reliable communication in internet environments.

TCP Segment
: The unit of data exchanged between TCP modules (including the TCP header).

Total Length
: The internet header field Total Length is the length of the datagram in octets including internet header and data.

Type of Service
: An internet header field which indicates the type (or quality) of service for this internet datagram.

User
: The user of the internet protocol. This may be a higher level protocol module, an application program, or a gateway program.

Version
: The Version field indicates the format of the internet header.

## REFERENCES

[1] Cerf, V., "The Catenet Model for Internetworking," Information Processing Techniques Office, Defense Advanced Research Projects Agency, IEN 48, July 1978.

[2] Bolt Beranek and Newman, "Specification for the Interconnection of a Host and an IMP," BBN Technical Report 1822, May 1978 (Revised).

[3] Shoch, J., "Inter-Network Naming, Addressing, and Routing," COMPCON, IEEE Computer Society, Fall 1978.

[4] Postel, J., "Address Mappings," IEN 115, USC/Information Sciences Institute, August 1979.

[5] Shoch, J., "Packet Fragmentation in Inter-Network Protocols," Computer Networks, v. 3, n. 1, February 1979.

[6] Postel, J., "Assigned Numbers," RFC 762, IEN 127, USC/Information Sciences Institute, January 1980.

# James Bond Meets The 7 Layer OSI Model

The modular networking architecture of Windows 95 is based on two industry standard models for a layered networking architecture, namely the International Organization for Standardization (ISO) model for computer networking, called the Open Systems Interconnect (OSI) Reference Model, and the Institute of Electrical and Electronic Engineers (IEEE) 802 model. Windows NT and Windows for Workgroups are also designed according to these standard models. The ISO OSI and IEEE 802 models define a modular approach to networking, with each layer responsible for some discrete aspect of the networking process.

The OSI model describes the flow of data in a network, from the lowest layer (the physical connections) up to the layer containing the user's applications. Data going to and from the network is passed layer to layer. Each layer is able to communicate with the layer immediately above it and the layer immediately below it. This way, each layer is written as an efficient, streamlined software component. When a layer receives a packet of information, it checks the destination address, and if its own address is not there, it passes the packet to the next layer.

When two computers communicate on a network, the software at each layer on one computer assumes it is communicating with the same layer on the other computer. For example, the Transport layer of one computer communicates with the Transport layer on the other computer. The Transport layer on the first computer has no regard for how the communication actually passes through the lower layers of the first computer, across the physical media, and then up through the lower layers of the second computer.

**The OSI Reference Model includes seven layers:**

☑ **Application**

☑ **Presentation**

☑ **Session**

☑ **Transport**

☑ **Network**

☑ **Data-Link**

☑ **Physical**

James Bond meets Number One on the 7th floor of the spy headquarters building. Number One gives Bond a secret message that must get through to the US Embassy across town. Bond proceeds to the 6th floor where the message is translated into an intermediary language, encrypted and miniaturized. Bond takes the elevator to the 5th

floor where Security checks the message to be sure it is all there and puts some checkpoints in the message so his counterpart at the US end can be sure he's got the whole message. On the 4th floor the message is analyzed to see if it can be combined with some other small messages that need to go to the US end. Also if the message was very large it might be broken into several small packages so other spies can take it and have it reassembled on the other end. The 3rd floor personnel check the address on the message and determine who the addressee is and advising Bond of the fastest route to the Embassy. On the 2nd floor the message is put into a special courier pouch(packet). It contains the message, the sender and destination ID. It also warns the recipient if other pieces are still coming. Bond proceeds to the 1st floor where Q has prepared the Aston Martin for the trip to the Embassy. Bond departs for the US Embassy with the secret packet in hand. On the other end the process is reversed. Bond proceeds from floor to floor where the message is decoded. The US Ambassador is very grateful the message got through safely. "Bond, please tell Number One I'll be glad to meet him for dinner tonight".

- The *Application layer* represents the level at which applications access network services. This layer represents the services that directly support applications such as software for file transfers, database access, and electronic mail.

- The *Presentation layer* translates data from the Application layer into an intermediary format. This layer also manages security issues by providing services such as data encryption, and compresses data so that fewer bits need to be transferred on the network.

- The *Session layer* allows two applications on different computers to establish, use, and end a session. This layer establishes dialog control between the two computers in a session, regulating which side transmits, plus when and how long it transmits.

- The *Transport layer* handles error recognition and recovery. It also repackages long messages when necessary into small packets for transmission and, at the receiving end, rebuilds packets into the original message. The receiving Transport layer also sends receipt acknowledgments.

- The *Network layer* addresses messages and translates logical addresses and names into physical addresses. It also determines the route from the source to the destination computer and manages traffic problems, such as switching, routing, and controlling the congestion of data packets.

- The *Data Link layer* packages raw bits from the Physical layer into frames (logical, structured packets for data). This layer is responsible for transferring frames from one computer to another, without errors. After sending a frame, it waits for an acknowledgment from the receiving computer.

- The *Physical layer* transmits bits from one computer to another and regulates the

transmission of a stream of bits over a physical medium. This layer defines how the cable is attached to the network adapter and what transmission technique is used to send data over the cable.

---

Date last updated: 05/06/97

# Protocol Stacks in Relationship to the OSI Model

| OSI Layer | Apple Computer | Banyan Systems | DEC DECnet | IBM SNA | Microsoft Networking | Novell NetWare | TCP/IP Internet | Xerox XNS | OSI Protocols |
|---|---|---|---|---|---|---|---|---|---|
| **Application** Layer 7 | Application Programs and Protocols for file transfer, electronic mail, etc. | | | | | | | | |
| **Presentation** Layer 6 | AppleTalk Filing Protocol (AFP) | Remote Procedural Calls (Net RPC) | Network Management Network Application | Transaction Services Presentation Services | Server Message Block (SMB) | NetWare Core Protocols (NCP | (Telnet, FTP, SMTP, etc.) | Control and Process Interaction | ISO 8823 |
| **Session** Layer 5 | AppleTalk Session Protocol (ASP) | | Session | Data Flow Control | Network Basic Input/Output System (NetBIOS) | Network Basic Input/Output System (NetBIOS) | | | ISO 8327 |
| **Transport** Layer 4 | AppleTalk Transaction Protocol (ATP) | VINES InterProcess Communications (VIPC) | End Communications | Transmission Control | Network Basic Extended User Interface (NetBEUI) | Sequenced Packet Exchange (SPX) | Transmission Control Protocol (TCP), Unacknowledged Datagram Protocol (UDP) | Sequenced Packet Protocol (SPP) | ISO 8073 TP0-4 |
| **Network** Layer 3 | Datagram Delivery Protocol (DDP) | VINES Internet Protocol (VIP) | Routing | Path Control | | Internet Packet Exchange (IPX) | Internet Protocol (IP) | Internet Datagram Protocol (IDP) | ISO 8473 (CLNP) |
| **Data Link** Layer 2 | Network Interface Cards: Ethernet, Token-Ring, ARCNET, StarLAN, LocalTalk, FDDI, ATM, etc. NIC Drivers: Open Datalink Interface (ODI), Network Independent Interface Specification (NDIS) | | | | | | | | |
| **Physical** Layer 1 | Transmission Media: Twisted Pair, Coax, Fiber Optic, Wireless Media, etc. | | | | | | | | |

*Return to Home*

# Inferring Internet Denial-of-Service Activity

Abstract for "Inferring Internet Denial-of-Service Activity" authored by David Moore, Geoffrey Voelker, and Stefan Savage. Published in proceedings of the 2001 USENIX Security Symposium.

Related animation: CAIDA's backscatter/denial-of-service animation available in quicktime and mpg format.

| Abstract Press Coverage Slides View full paper:
PDF gzipped postscript |

## Inferring Internet Denial-of-Service Activity

**David Moore**
**Cooperative Association for Internet Data Analysis (CAIDA)**
**San Diego Supercomputer Center**
**University of California, San Diego**

**Geoffrey M. Voelker and Stefan Savage**
**Department of Computer Science and Engineering**
**University of California, San Diego**

In this paper, we seek to answer a simple question: "How prevalent are denial-of-service attacks in the Internet today?". Our motivation is to understand quantitatively the nature of the current threat as well as to enable longer-term analyses of trends and recurring patterns of attacks. We present a new technique, called "backscatter analysis", that provides an estimate of worldwide denial-of-service activity. We use this approach on three week-long datasets to assess the number, duration and focus of attacks, and to characterize their behavior. During this period, we observe more than 12,000 attacks against more than 5,000 distinct targets, ranging from well known e-commerce companies such as Amazon and Hotmail to small foreign ISPs and dial-up connections. We believe that our work is the only publically available data quantifying denial-of-service activity in the Internet.

| Abstract Press Coverage Slides View full paper:

Page URL: http://www.caida.org/outreach/papers/2001/BackScatter/index.xml
Last updated: Sat Jan 12 06:23:33 PST 2002
Maintained by: Alex Ma

# Inferring Internet Denial-of-Service Activity

David Moore
*CAIDA*
*San Diego Supercomputer Center*
*University of California, San Diego*
dmoore@caida.org

Geoffrey M. Voelker and Stefan Savage
*Department of Computer Science and Engineering*
*University of California, San Diego*
{voelker,savage}@cs.ucsd.edu

## Abstract

In this paper, we seek to answer a simple question: "How prevalent are denial-of-service attacks in the Internet today?". Our motivation is to understand quantitatively the nature of the current threat as well as to enable longer-term analyses of trends and recurring patterns of attacks. We present a new technique, called "backscatter analysis", that provides an estimate of *worldwide* denial-of-service activity. We use this approach on three week-long datasets to assess the number, duration and focus of attacks, and to characterize their behavior. During this period, we observe more than 12,000 attacks against more than 5,000 distinct targets, ranging from well known e-commerce companies such as Amazon and Hotmail to small foreign ISPs and dial-up connections. We believe that our work is the only publically available data quantifying denial-of-service activity in the Internet.

## 1 Introduction

In February of 2000, a series of massive denial-of-service (DoS) attacks incapacitated several high-visibility Internet e-commerce sites, including Yahoo, Ebay, and E*trade. Next, in January of 2001, Microsoft's name server infrastructure was disabled by a similar assault. Despite attacks on high-profile sites, the majority of attacks are not well publicized. Many other domestic and foreign sites have also been victims, ranging from smaller commercial sites, to educational institutions, public chat servers and government organizations.

While it is clear from these anecdotal reports that denial-of-service attacks continue to be a problem, there is currently not much quantitative data about the prevalence of these attacks nor any representative characterization of their behavior. Unfortunately, there are multiple obstacles hampering the collection of an authoritative denial-of-service traffic dataset. Service providers and content providers consider such data sensitive and private. Even if it were allowed, monitoring traffic at enough sites to obtain a representative measure of Internet-wide attacks presents a significant logistical challenge. Consequently, the only contemporary public data we are aware of is a CSI/FBI survey study [8][1].

We believe that a strong quantitative foundation is necessary both for understanding the nature of today's threat and as a baseline for longer-term comparison and analysis. Our paper seeks to answer the simple question: "How prevalent are denial-of-service attacks in the Internet today?". As a means to this end, we describe a traffic monitoring technique called "backscatter analysis" for estimating the *worldwide* prevalence of denial-of-service attacks. Using backscatter analysis, we observe 12,805 attacks on over 5,000 distinct Internet hosts belonging to more than 2,000 distinct organizations during a three-week period. We further are able to estimate a lower-bound on the intensity of such attacks – some of which are in excess of 600,000 packets-per-second (pps) – and characterize the nature of the sites victimized.

The remainder of this paper is organized as follows: Section 2 describes the underlying mechanisms of denial-of-service attacks, Section 3 describes the backscatter technique, and limitations arising from its assumptions, and Section 4 explains our techniques for classifying attacks from monitored backscatter traffic. In Section 5 we describe our experimental platform, and present our results in Section 6. Finally, in Sections 7 and 8 we cover related work and summarize our find-

[1] The primary result from this report is that 27 percent of security professionals surveyed detected denial-of-service attacks during the year 2000.

ings.

## 2 Background

Denial-of-service attacks consume the resources of a remote host or network that would otherwise be used for serving legitimate users. There are two principal classes of attacks: *logic* attacks and *flooding* attacks. Attacks in the first class, such as the "Ping-of-Death", exploit existing software flaws to cause remote servers to crash or substantially degrade in performance. Many of these attacks can be prevented by either upgrading faulty software or filtering particular packet sequences, but they remain a serious and ongoing threat. The second class, flooding attacks, overwhelm the victim's CPU, memory, or network resources by sending large numbers of spurious requests. Because there is typically no simple way to distinguish the "good" requests from the "bad", it can be extremely difficult to defend against flooding attacks. For the purposes of this study we will focus solely on flooding attacks.

### 2.1 Attack types

There are two related consequences to a flooding attack – the network load induced and the impact on the victim's CPU. To load the network, an attacker generally sends small packets as rapidly as possible since most network devices (both routers and NICs) are limited not by bandwidth but by packet processing rate. Therefore, packets-per-second are usually the best measure of network load during an attack.

An attacker often simultaneously attempts to load the victim's CPU by requiring additional processing above and beyond that required to receive a packet. For example, the best known denial-of-service attack is the "SYN flood" [6] which consists of a stream of TCP SYN packets directed to a listening TCP port at the victim. For each such SYN packet received, the host victim must search through existing connections and if no match is found, allocate a new data structure for the connection. Moreover, the number of these data structures may be limited by the victim's operating system. Consequently, without additional protection, even a small SYN flood can overwhelm a remote host. There are many similar attacks that exploit other code vulnerabilities including TCP ACK, NUL, RST and DATA floods, IP fragment floods, ICMP Echo Request floods, DNS Request floods, and so forth.

### 2.2 Distributed attacks

While a single host can cause significant damage by sending packets at its maximum rate, attackers can (and do) mount more powerful attacks by leveraging the resources of multiple hosts. Typically an attacker compromises a set of Internet hosts (using manual or semi-automated methods) and installs a small attack daemon on each, producing a group of "zombie" hosts. This daemon typically contains both the code for sourcing a variety of attacks and some basic communications infrastructure to allow for remote control. Using variants of this basic architecture an attacker can focus a coordinated attack from thousands of zombies onto a single site.

| Packet sent | Response from victim |
|---|---|
| TCP SYN (to open port) | TCP SYN/ACK |
| TCP SYN (to closed port) | TCP RST (ACK) |
| TCP ACK | TCP RST (ACK) |
| TCP DATA | TCP RST (ACK) |
| TCP RST | no response |
| TCP NULL | TCP RST (ACK) |
| ICMP ECHO Request | ICMP Echo Reply |
| ICMP TS Request | ICMP TS Reply |
| UDP pkt (to open port) | protocol dependent |
| UDP pkt (to closed port) | ICMP Port Unreach |
| ... | ... |

Table 1: A sample of victim responses to typical attacks.

### 2.3 IP spoofing

To conceal their location, thereby forestalling an effective response, attackers typically forge, or "spoof", the IP source address of each packet they send. Consequently, the packets appear to the victim to be arriving from one or more third parties. Spoofing can also be used to "reflect" an attack through an innocent third party. While we do not address "reflector attacks" in this paper, we describe them more fully in Section 3.3.

## 3 Basic methodology

As noted in the previous section, attackers commonly spoof the source IP address field to conceal the location of the attacking host. The key observation behind our technique is that for direct denial-of-service attacks, most programs select source addresses at random for each packet sent. These programs include all of the most popular distributed attacking tools: Shaft, TFN, TFN2k, trinoo, all variants of Stacheldraht, mstream and Trinity). When a spoofed packet arrives at the victim, the victim usually sends what it believes to be an appropriate response to the faked IP address (such as shown in Table 1). Occasionally, an intermediate network device (such as a router, load balancer, or firewall) may issue its own reply to the attack via an ICMP message [21].

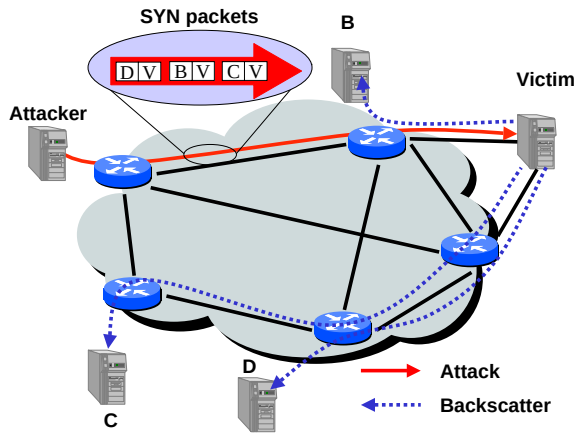



Figure 1: An illustration of backscatter in action. Here the attacker sends a series of SYN packets towards the victim V, using a series of random spoofed source addresses: named C, B, and D. Upon receiving these packets the victim responds by sending SYN/ACKs to each of spoofed hosts.

Again, these ICMP messages are sent to the randomly spoofed source address.

Because the attacker's source address is selected at random, the victim's responses are equi-probably distributed across the entire Internet address space, an inadvertent effect we call "backscatter"[2]. This behavior is illustrated in Figure 1.

### 3.1 Backscatter analysis

Assuming per-packet random source addresses, reliable delivery and one response generated for every packet in an attack, the probability of a given host on the Internet receiving at least one unsolicited response from the victim is $\frac{m}{2^{32}}$ during an attack of $m$ packets. Similarly, if one monitors $n$ distinct IP addresses, then the expectation of observing an attack is:

$$E(X) = \frac{nm}{2^{32}}$$

By observing a large enough address range we can effectively "sample" all such denial-of-service activity on the Internet. Contained in these samples are the identity of the victim, information about the kind of attack, and a timestamp from which we can estimate attack duration. Moreover, given these assumptions, we can also use the average arrival rate of unsolicited responses directed at the monitored address range to estimate the actual rate of the attack being directed at the victim, as follows:

$$R \geq R' \frac{2^{32}}{n}$$

where $R'$ is the measured average inter-arrival rate of backscatter from the victim and $R$ is the extrapolated attack rate in packets-per-second.

### 3.2 Address uniformity

The estimation approach outlined above depends on the spoofed source addresses being uniformly distributed across the entire IP address space. To check whether a sample of observed addresses are uniform in our monitored address range, we compute the Anderson-Darling (A2) test statistic [9] to determine if the observations are consistent with a uniform distribution. In particular, we use the implementation of the A2 test as specified in RFC2330 [19] at a 0.05 significance level.

### 3.3 Analysis limitations

There are three assumptions that underly our analysis:

- *Address uniformity*: attackers spoof source addresses at random.
- *Reliable delivery*: attack traffic is delivered reliably to the victim and backscatter is delivered reliably to the monitor.
- *Backscatter hypothesis*: unsolicited packets observed by the monitor represent backscatter.

We discuss potential biases that arise from these assumptions below.

Key among our assumptions is the random selection of source address. There are three reasons why this assumption may not be valid. First, some ISPs employ *ingress filtering* [12, 5] on their routers to drop packets with source IP addresses outside the range of a customer's network. Thus, an attacker's source address range may not include any of our monitored addresses and we will underestimate the total number of attacks.

"Reflector attacks" pose a second problem for source address uniformity. In this situation, an attacker "launders" the attack by sending a packet spoofed with the victim's source address to a third party. The third party responds by sending a response back towards the victim. If the packets to the third partie are addressed using a broadcast address (as with the popular smurf or fraggle attacks) then third parties may further amplify the attack. The key issue with reflector attacks is that the source address is specifically selected. Unless an IP address in the range we monitor is used as a reflector, we will be unable

[2]We did not originate this term. It is borrowed from Vern Paxson who independently discovered the same backscatter effect when an attack accidentally disrupted multicast connectivity by selecting global multicast addresses as source addresses [20].

to observe the attack. We have detected no instances of a monitored host involved in this sort of attack. Our inability to detect, "reflector attacks" cause us to underestimate the total number of denial-of-service attacks.

Finally, if the distribution of source addresses is not random, then any attempt to extrapolate the attack rate via the arrival rate of responses will produce an arbitrarily biased result. This particular problem can be mitigated by verifying that the distribution of observed source addresses is indeed uniform within the set of $n$ addresses we observe.

Another limitation arises from our assumption that packets are delivered reliably and that every packet generates a response. During a large attack it is likely that packets from the attacker may be queued and dropped. Those packets that *do* arrive may be filtered or rate-limited by firewall or intrusion detection software [4] and moreover some forms of attack traffic (e.g., TCP RST messages) do not typically elicit a response. Finally, the responses themselves may be queued and dropped along the path back to our monitored address range. In particular, our estimate of the attack rate is necessarily limited to the capacity of smallest bottleneck link between the victim and our monitor. As with our random distribution assumption, these limitations will cause us to *underestimate* the number of attacks and the attack rate. However, they may also bias our characterization of victims (e.g., if large e-commerce sites are more likely to have rate-limiting software than educational sites, then we may disproportionately underestimate the size of attacks on this class of victim).

The final limitation of our technique is that we assume unsolicited responses represent backscatter from an attack. Any server on the Internet is free to send unsolicited packets to our monitored addresses, and these packets may be misinterpreted as backscatter from an attack. It is possible to eliminate accidental errors by choosing a quiescent address range for monitoring, filtering those packet flows consistently destined to a single host in the range and by high-pass filtering to only record sufficiently long and voluminous packet flows. However, a concerted effort by a third-party to bias our results would be difficult to detect and correct automatically. The most likely source of such bias arises from misinterpretation of random port scans as backscatter. While it is impossible to eliminate this possibility in general, we will show that it is extremely unlikely to be a factor in the vast majority of attacks we observe.

In spite of its limitations, we believe our overall approach is sound and provides at worst a conservative estimate of current denial-of-service activity.

# 4 Attack Classification

After collecting a large trace of backscatter packets, the first task is post-processing the trace. For this we group collections of related packets into clusters representing attacks. The choice of a specific aggregation methodology presents significant challenges. For example, it is often unclear whether contemporaneous backscatter indicating both TCP and ICMP-based attacks should be classified as a single attack or multiple attacks. More difficult still is the problem of determining the start and end times of an attack. In the presence of significant variability, too lenient a threshold can bias the analysis towards fewer attacks of longer duration and low average packet rates, while too strict an interpretation suggests a large number of short attacks with highly variable rates.

Without knowledge of the intent of the attacker or direct observation of the attack as it orchestrated by the attacker, it is impossible to create a synthetic classification system that will group all types of attacks appropriately for all metrics. Instead, we have chosen to employ two distinct classification methods: a flow-based analysis for classifying individual attacks – how many, how long and what kind – and an event-based method for analyzing the severity of attacks on short time scales.

## 4.1 Flow-based classification

For the purpose of this study, we define a flow as a series of consecutive packets sharing the same target IP address and IP protocol. We explored several approaches for defining flow lifetimes and settled on a fixed timeout approach: the first packet seen for a target creates a new flow and any additional packets from that target are counted as belonging to that flow if the packets are received within five minutes of the most recent packet in this flow. The choice of parameters here can influence the final results, since a more conservative timeout will tend to suggest fewer, longer attacks, while a shorter timeout will suggest a large number of short attacks. We chose five minutes as a human-sensible balance that is not unduly affected by punctuated attacks or temporary outages.

To reduce noise and traffic generated due to random Internet misconfiguration (for instance, one NetBIOS implementation/configuration sends small numbers unsolicited packets to our monitored address range) we discard all flows that do not have at least 100 packets and a flow duration of at least 60 seconds. These parameters are also somewhat arbitrary, but we believe they represent a reasonable baseline – below such thresholds it seems unlikely that an attack would cause significant damage. Finally, flows must contain packets sent to more than one of our monitored addresses.

We examine each individual flow and extract the following information:

- *TCP flag settings*: whether the flow consists of SYN/ACKs, RSTs, etc.
- *ICMP payload*: for ICMP packets that contain copies of the original packet (e.g. TTL expired) we break out the enclosed addresses, protocols, ports, etc.
- *Address uniformity*: whether the distribution of source addresses within our monitored range passes the Anderson-Darling (A2) test for uniformity to the 0.05 significance level.
- *Port settings*: for source and destination ports (for both UDP and TCP) we record whether the port range is fixed, is uniform under the A2 test, or is non-fixed and non-uniform.
- *DNS information*: the full DNS address of the source address – the victim.
- *Routing information*: the prefix, mask and origin AS as registered in our local BGP table on the morning of February 7th.

We generate a database in which each record characterizes the properties of a single attack.

### 4.2 Event-based classification

Because the choice of flow parameters can impact the estimated duration of an attack, the flow-based method may obscure interesting time-domain characteristics. In particular, attacks can be highly variable – with periodic bursts of activity – causing the flow-based method to vastly underestimate the short-term impact of an attack and overestimate the long-term impact.

We use an event-based classification method keyed entirely on the victim's IP address over fixed time-windows for examining time-domain qualities, such as the number of simultaneous attacks or the distribution of attack rates, For these analyses we divide our trace into one minute periods and record each *attack event* during this period. An attack event is defined by a victim emitting at least ten backscatter packets during a one minute period. We do not further classify attacks according to protocol type, port, etc, as the goal is to estimate the instantaneous impact on a particular victim. The result of this classification is a database in which each record characterizes the number of victims and the intensity of the attacks in each one minute period.

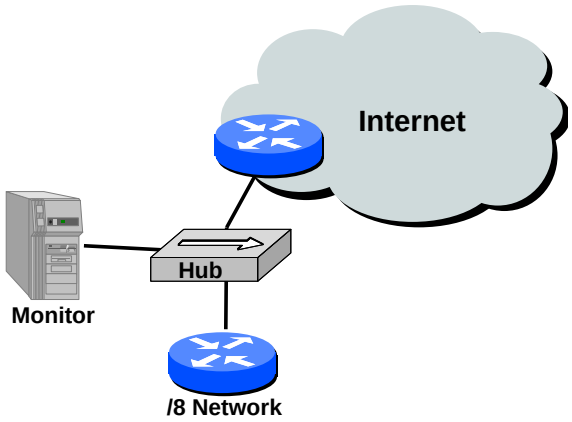



Figure 2: Our experimental backscatter collection platform. We monitor all traffic to our /8 network by passively monitoring data as it is forwarded through a shared hub. This monitoring point represents the only ingress into the network.

## 5 Experimental platform

For our experiments monitored the sole ingress link into a lightly utilized /8 network (comprising $2^{24}$ distinct IP addresses, or 1/256 of the total Internet address space). Our monitoring infrastructure, shown in Figure 2, consisted of a PC configured to capture all Ethernet traffic, attached to a shared hub at the router terminating this network. During this time, the upstream router did filter some traffic destined to the network (notably external SNMP queries) but we do not believe that this significantly impacted our results. We also have some evidence that small portions of our address prefix are occasionally "hijacked" by inadvertent route advertisements elsewhere in the Internet, but at worst this should cause us to slightly underestimate attack intensities. We collected three traces, each roughly spanning one week, starting on February 1st and extending to February 25th, and isolated the inbound portion of the network.

## 6 Results

Using the previously described flows-based approach (Section 4.1), we observed 12,805 attacks over the course of a week. Table 2 summarizes this data, showing more than 5,000 distinct victim IP addresses in more than 2,000 distinct DNS domains. Across the entire period we observed almost 200 million backscatter packets (again, representing less than $\frac{1}{256}$ of the actual attack traffic during this period).

In this section, we first show the overall frequency of attacks seen in our trace, and then characterize the attacks according to both the type of attack and the type of victim.

| | Trace-1 | Trace-2 | Trace-3 |
|---|---|---|---|
| Dates (2001) | Feb 01 – 08 | Feb 11 – 18 | Feb 18 – 25 |
| Duration | 7.5 days | 6.2 days | 7.1 days |
| Flow-based Attacks: | | | |
| Unique victim IPs | 1,942 | 1,821 | 2,385 |
| Unique victim DNS domains | 750 | 693 | 876 |
| Unique victim DNS TLDs | 60 | 62 | 71 |
| Unique victim network prefixes | 1,132 | 1,085 | 1,281 |
| Unique victim Autonomous Systems | 585 | 575 | 677 |
| Attacks | 4,173 | 3,878 | 4,754 |
| Total attack packets | 50,827,217 | 78,234,768 | 62,233,762 |
| Event-based Attacks: | | | |
| Unique victim IPs | 3,147 | 3,034 | 3,849 |
| Unique victim DNS domains | 987 | 925 | 1,128 |
| Unique victim DNS TLDs | 73 | 71 | 81 |
| Unique victim network prefixes | 1,577 | 1,511 | 1,744 |
| Unique victim Autonomous Systems | 752 | 755 | 874 |
| Attack Events | 112,457 | 102,204 | 110,025 |
| Total attack packets | 51,119,549 | 78,655,631 | 62,394,290 |

Table 2: Summary of backscatter database.

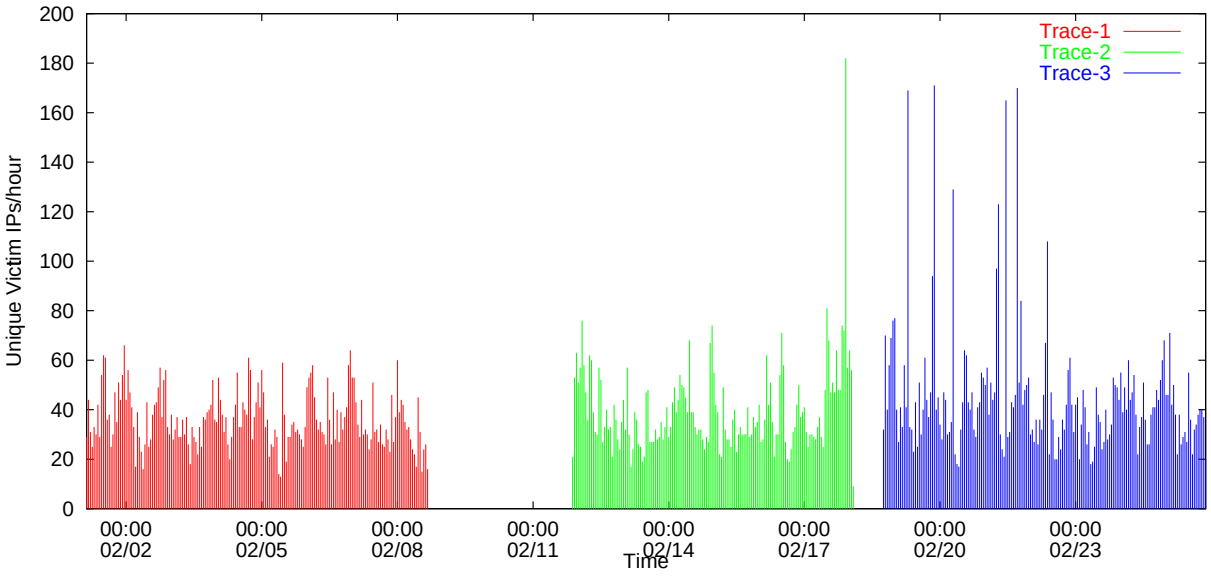



Figure 3: Estimated number of attacks per hour as a function of time (UTC).

| Kind | Trace-1 | | Trace-2 | | Trace-3 | |
|---|---|---|---|---|---|---|
| | Attacks | Packets (k) | Attacks | Packets (k) | Attacks | Packets (k) |
| TCP (RST ACK) | 2,027 (49) | 12,656 (25) | 1,837 (47) | 15,265 (20) | 2,118 (45) | 11,244 (18) |
| ICMP (Host Unreachable) | 699 (17) | 2,892 (5.7) | 560 (14) | 27,776 (36) | 776 (16) | 19,719 (32) |
| ICMP (TTL Exceeded) | 453 (11) | 31,468 (62) | 495 (13) | 32,001 (41) | 626 (13) | 22,150 (36) |
| ICMP (Other) | 486 (12) | 580 (1.1) | 441 (11) | 640 (0.82) | 520 (11) | 472 (0.76) |
| TCP (SYN ACK) | 378 (9.1) | 919 (1.8) | 276 (7.1) | 1,580 (2.0) | 346 (7.3) | 937 (1.5) |
| TCP (RST) | 128 (3.1) | 2,309 (4.5) | 269 (6.9) | 974 (1.2) | 367 (7.7) | 7,712 (12) |
| TCP (Other) | 2 (0.05) | 3 (0.01) | 0 (0.00) | 0 (0.00) | 1 (0.02) | 0 (0.00) |

Table 3: Breakdown of response protocols.

## 6.1 Time series

Figure 3 shows a time series graph of the estimated number of actively attacked victims throughout the three traces, as sampled in one hour periods. There are two gaps in this graph corresponding to the gaps between traces. In contrast to other workloads, such as HTTP, the number of active attacks does not appear to follow any diurnal pattern (at least as observed from a single location). The outliers on the week of February 20th, with more than 150 victim IP addresses per hour, represent broad attacks against many machines in a common network. While most of the backscatter data averages one victim IP address per network prefix per hour, the ratio climbs to above five for many outliers.

## 6.2 Attack classification

In this section we characterize attacks according to the protocols used in response packets sent by victims, the protocols used in the original attack packets, and the rate and durations of attacks.

### 6.2.1 Response protocols

In Table 3 we decompose our backscatter data according to the protocols of responses returned by the victim or an intermediate host. For each trace we list both the number of attacks and the number backscatter packets for the given protocol. The numbers in parentheses show the relative percentage represented by each count. For example, 1,837 attacks in Trace 2 (47% of the total), were derived from TCP backscatter with the RST and ACK flags set.

We observe that over 50% of the attacks and 20% of the backscatter packets are TCP packets with the RST flag set. Referring back to Table 1 we see that RST is sent in response to either a SYN flood directed against a closed port or some other unexpected TCP packet. The next largest protocol category is ICMP host unreachable, comprising roughly 15% of the attacks. Almost all of these ICMP messages contain the TCP header from a packet directed at the victim, suggesting a TCP flood of some sort. Unfortunately, the TCP flags field cannot be recovered, because the ICMP response only includes the first 28 bytes of the original IP packet. ICMP host unreachable is generally returned by a router when a packet cannot be forwarded to its destination. Probing some of these victims we confirmed that a number of them could not be reached, but most were accessible, suggesting intermittent connectivity. This discontinuous reachability is probably caused by explicit "black holing' on the part of an ISP.

We also see a number of SYN/ACK backscatter packets (likely sent directly in response to a SYN flood on an open port) and an equivalent number of assorted ICMP messages, including ICMP echo reply (resulting from ICMP echo request floods), ICMP protocol unreachable (sent in response to attacks using illegal combinations of TCP flags), ICMP fragmentation needed (caused by attacks with the "Dont Fragment" bit set) and ICMP administratively filtered (likely the result of some attack countermeasure). However, a more surprising finding is the large number of ICMP TTL exceeded messages – comprising between 36% and 62% of all backscatter packets observed, yet less than 15% of the total attacks. In fact, the vast majority of these packets occur in just a few attacks, including three attacks on @Home customers, two on China Telecom (one with almost 9 million backscatter packets), and others directed at Romania, Belgium, Switzerland and New Zealand. The attack on the latter was at an extremely high rate, suggesting an attack of more than 150,000 packets per second. We are unable to completely explain the mechanism for the generation of these time-exceeded messages. Upon examination of the encapsulated header that is returned, we note that several of them share identical "signatures" (ICMP Echo with identical sequence number, identification fields, and checksum) suggesting that a single attack tool was in use.

### 6.2.2 Attack protocols

We refine this data in Table 4 to show the distribution of *attack protocols*. That is, the protocol which must

| Kind | Trace-1 | | Trace-2 | | Trace-3 | |
|---|---|---|---|---|---|---|
| | Attacks | Packets (k) | Attacks | Packets (k) | Attacks | Packets (k) |
| TCP | 3,902 (94) | 28,705 (56) | 3,472 (90) | 53,999 (69) | 4,378 (92) | 43,555 (70) |
| UDP | 99 (2.4) | 66 (0.13) | 194 (5.0) | 316 (0.40) | 131 (2.8) | 91 (0.15) |
| ICMP | 88 (2.1) | 22,020 (43) | 102 (2.6) | 23,875 (31) | 107 (2.3) | 18,487 (30) |
| Proto 0 | 65 (1.6) | 25 (0.05) | 108 (2.8) | 43 (0.06) | 104 (2.2) | 49 (0.08) |
| Other | 19 (0.46) | 12 (0.02) | 2 (0.05) | 1 (0.00) | 34 (0.72) | 52 (0.08) |

Table 4: Breakdown of protocols used in attacks.

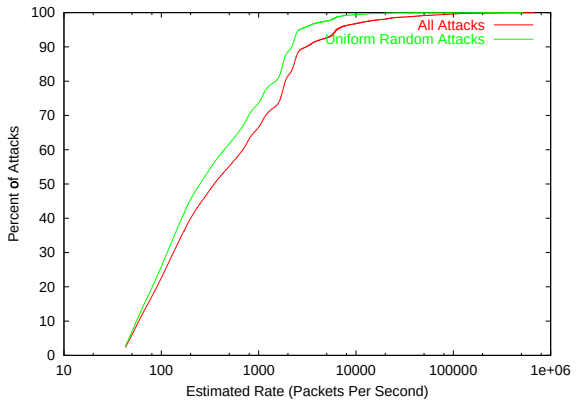



Figure 4: Cumulative distributions of estimated attack rates in packets per second.

have been used by the attacker to produce the backscatter monitored at our network. We see that more than 90% of the attacks use TCP as their protocol of choice, but a smaller number of ICMP-based attacks produce a disproportionate number of the backscatter packets seen. Other protocols represent a minor number of both attacks and backscatter packets. This pattern is consistent across all three traces.

In Table 5 we further break down our dataset based on the service (as revealed in the victim's port number) being attacked. Most of the attacks focus on multiple ports, rather than a single one and most of these are well spread throughout the address range. Many attack programs select random ports above 1024; this may explain why less than 25% of attacks show a completely uniform random port distribution according to the A2 test. Of the remaining attacks, the most popular static categories are port 6667 (IRC), port 80 (HTTP), port 23 (Telnet), port 113 (Authd). The large number of packets directed at port 0 is an artifact of our ICMP categorization – there are fewer than ten TCP attacks directed at port 0, comprising a total of less than 9,000 packets.

### 6.2.3 Attack rate

Figure 4 shows two cumulative distributions of attack event rates in packets per second. The lower curve shows the cumulative distribution of event rates for all attacks, and the upper curve shows the cumulative distribution of event rates for uniform random attacks, i.e., those attacks whose source IP addresses satisfied the A2 uniform distribution test described in Section 3.2. As described earlier, we calculate the attack event rate by multiplying the average arrival rate of backscatter packets by 256 (assuming that an attack represents a random sampling across the entire address space, of which we monitor $\frac{1}{256}$). Almost all attacks have no dominant mode in the address distribution, but sometimes small deviations from uniformity prevent the A2 test from being satisfied. For this reason we believe that there is likely some validity in the extrapolation applied to the complete attack dataset. Note that the attack rate (x-axis) is shown using a logarithmic scale.

Comparing the distributions, we see that the uniform random attacks have a lower rate than the distribution of all attacks, but track closely. Half of the uniform random attack events have a packet rate greater than 250, whereas half of all attack events have a packet rate greater than 350. The fastest uniform random event is over 517,000 packets per second, whereas the fastest overall event is over 679,000 packets per second.

How threatening are the attacks that we see? Recent experiments with SYN attacks on commercial platforms show that an attack rate of only 500 SYN packets per second is enough to overwhelm a server [10]. In our trace, 38% of uniform random attack events and 46% of all attack events had an estimated rate of 500 packets per second or higher. The same experiments show that even with a specialized firewall designed to resist SYN floods, a server can be disabled by a flood of 14,000 packets per second. In our data, 0.3% of the uniform random attacks and 2.4% of all attack events would still compromise these attack-resistant firewalls. We conclude that the majority of the attacks that we have monitored are fast enough to overwhelm commodity solutions, and a small fraction are fast enough to overwhelm even optimized countermeasures.

Of course, one significant factor in the question of threat posed by an attack is the connectivity of the victim. An attack rate that overwhelms a cable modem victim may be trivial a well-connected major server installation. Victim connectivity is a difficult to ascertain with-

| Kind | Trace-1 | | Trace-2 | | Trace-3 | |
|---|---|---|---|---|---|---|
| | Attacks | Packets (k) | Attacks | Packets (k) | Attacks | Packets (k) |
| Multiple Ports | 2,740 (66) | 24,996 (49) | 2,546 (66) | 45,660 (58) | 2,803 (59) | 26,202 (42) |
| Uniformly Random | 655 (16) | 1,584 (3.1) | 721 (19) | 5,586 (7.1) | 1,076 (23) | 15,004 (24) |
| Other | 267 (6.4) | 994 (2.0) | 204 (5.3) | 1,080 (1.4) | 266 (5.6) | 410 (0.66) |
| Port Unknown | 91 (2.2) | 44 (0.09) | 114 (2.9) | 47 (0.06) | 155 (3.3) | 150 (0.24) |
| HTTP (80) | 94 (2.3) | 334 (0.66) | 79 (2.0) | 857 (1.1) | 175 (3.7) | 478 (0.77) |
| 0 | 78 (1.9) | 22,007 (43) | 90 (2.3) | 23,765 (30) | 99 (2.1) | 18,227 (29) |
| IRC (6667) | 114 (2.7) | 526 (1.0) | 39 (1.0) | 211 (0.27) | 57 (1.2) | 1,016 (1.6) |
| Authd (113) | 34 (0.81) | 49 (0.10) | 52 (1.3) | 161 (0.21) | 53 (1.1) | 533 (0.86) |
| Telnet (23) | 67 (1.6) | 252 (0.50) | 18 (0.46) | 467 (0.60) | 27 (0.57) | 160 (0.26) |
| DNS (53) | 30 (0.72) | 39 (0.08) | 3 (0.08) | 3 (0.00) | 25 (0.53) | 38 (0.06) |
| SSH (22) | 3 (0.07) | 2 (0.00) | 12 (0.31) | 397 (0.51) | 18 (0.38) | 15 (0.02) |

Table 5: Breakdown of attacks by victim port number.

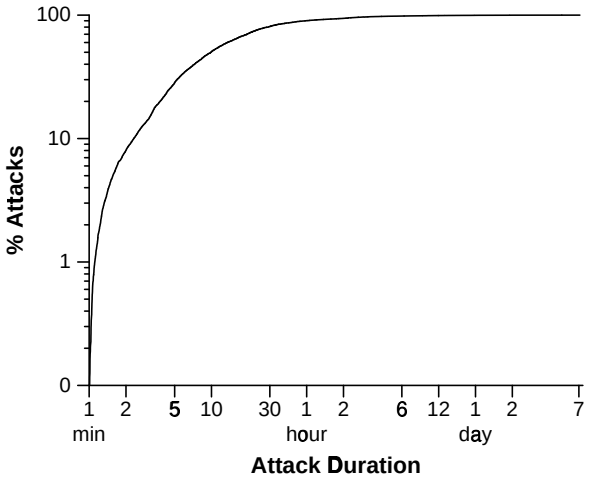



Figure 5: Cumulative distribution of attack durations.

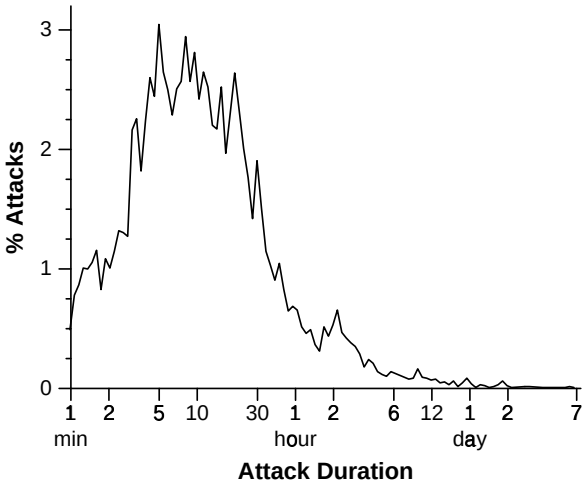



Figure 6: Probability density of attack durations.

out flooding the victim's link. Consequently, we leave correlation between attack rates and victim connectivity as an open problem.

### 6.2.4 Attack duration

While attack event rates characterize the intensity of attacks, they do not give insight on how long attacks are sustained. For this metric, we characterize the duration of attacks in Figures 5 and 6 across all three weeks of trace data. In these graphs, we use the flow-based classification described in Section 4 because flows better characterize attack durations while remaining insensitive to intensity. We also combine all three weeks of attacks for clarity; the distributions are nearly dentical for each week, and individual weekly curves overlap and obscure each other.

Figure 5 shows the cumulative distribution of attack durations in units of time; note that both the axes are logarithmic scale. In this graph we see that most attacks are relatively short: 50% of attacks are less than 10 minutes in duration, 80% are less than 30 minutes, and 90% last less than an hour. However, the tail of the distribution is long: 2% of attacks are greater than 5 hours, 1% are greater than 10 hours, and dozens spanned multiple days.

Figure 6 shows the probability density of attack durations as defined using a histogram of 150 buckets in the log time domain. The x-axis is in logarithmic units of time, and the y-axis is the percentage of attacks that lasted a given amount of time. For example, when the curve crosses the y-axis, it indicates that approximately 0.5% of attacks had a duration of 1 minute. As we saw in the CDF, the bulk of the attacks are relatively short, lasting from 3–20 minutes. From this graph, though, we see that there are peaks at rounded time durations in this interval at durations of 5, 10, and 20 minutes. Immediately before this interval there is a peak at 3 minutes, and immediately after a peak at 30 minutes. For attacks with longer durations, we see a local peak at 2 hours in the long tail.

## 6.3 Victim classification

In this section we characterize victims according to DNS name, top-level domain, Autonomous System, and degree of repeated attacks.

### 6.3.1 Victim Name

Table 6 shows the distribution of attacks according to the DNS name associated with the victim's IP address. We classify these using a hand-tuned set of regular expression matches (i.e. DNS names with "dialup" represent modems, "dsl" or "home.com" represent broadband, etc). The majority of attacks are not classified by this scheme, either because they are not matched by our criteria (shown by "other"), or more likely, because there was no valid reverse DNS mapping (shown by "In-Addr Arpa").

Of the remaining attacks there are several interesting observations. First, there is a significant fraction of attacks directed against home machines – either dialup or broadband. Some of these attacks, particularly those directed towards cable modem users, constitute relatively large, severe attacks with rates in the thousands of packets per second. This suggests that minor denial-of-service attacks are frequently being used to settle personal vendettas. In the same vein we anecdotally observe a significant number of attacks against victims running "Internet Relay Chat" (IRC), victims supporting multi-player game use (e.g. battle.net), and victims with DNS names that are sexually suggestive or incorporate themes of drug use. We further note that many reverse DNS mappings have been clearly been compromised by attackers (e.g., DNS translations such as "is.on.the.net.illegal.ly" and "the.feds.cant.secure.their.shellz.ca").

Second, there is a small but significant fraction of attacks directed against network infrastructure. Between 2–3% of attacks target name servers (e.g., ns4.reliablehosting.com), while 1–3% target routers (e.g., core2-corel-oc48.paol.above.net). Again, some of these attacks, particularly a few destined towards routers, are comprised of a disproportionately large number of packets. This point is particularly disturbing, since overwhelming a router could deny service to *all* end hosts that rely upon that router for connectivity.

Finally, we are surprised at the diversity of different commercial attack targets. While we certainly find attacks on bellwether Internet sites including aol.com, akamai.com, amazon.com and hotmail.com, we also see attacks against a large range of smaller and medium sized businesses.

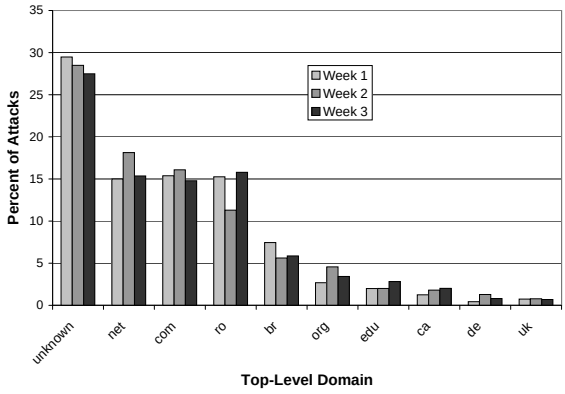



Figure 7: Distribution of attacks to the 10 top-level domains (TLDs) that received the most number of attacks.

### 6.3.2 Top-level domains

Figure 7 shows the distribution of attacks to the 10 most frequently targeted top-level domains (TLDs). For each TLD displayed on the x-axis, we show one value for each of the three weeks of our study in progressive shades of grey. Note that the TLDs are sorted by overall attacks across all three weeks.

Comparing the number of attacks to each TLD from week to week, we see that there is little variation. Each TLD is targeted by roughly the same percentage of attacks each week. The domain `unknown` represents those attacks in which a reverse DNS lookup failed on the victim IP address (just under 30% of all attacks). In terms of the "three-letter" domains, both `com` and `net` were each targeted by roughly 15% of the attacks, but `edu` and `org` were only targeted by 2–4% of the attacks. This is not surprising, as sites in the `com` and `net` present more attractive and newsworthy targets. Interestingly, although one might have expected attacks to sites in `mil`, `mil` did not show up in any of our reverse DNS lookups. We do not yet know what to conclude from this result; for example, it could be that `mil` targets fall into our `unknown` category.

In terms of the country-code TLDs, we see that there is a disproportionate concentration of attacks to a small group of countries. Surprisingly, Romania (`ro`), a country with a relatively poor networking infrastructure, was targeted nearly as frequently as `net` and `com`, and Brazil (`br`) was targeted almost more than `edu` and `org` combined. Canada, Germany, and the United Kingdom were all were targeted by 1–2% of attacks.

### 6.3.3 Autonomous Systems

As another aggregation of attack targets, we examined the distribution of attacks to Autonomous Systems (ASes). To determine the origin AS number associated

| Kind | Trace-1 | | | | Trace-2 | | | | Trace-3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Attacks | | Packets (k) | | Attacks | | Packets (k) | | Attacks | | Packets (k) | |
| Other | 1,917 | (46) | 19,118 | (38) | 1,985 | (51) | 25,305 | (32) | 2,308 | (49) | 17,192 | (28) |
| In-Addr Arpa | 1,230 | (29) | 16,716 | (33) | 1,105 | (28) | 24,645 | (32) | 1,307 | (27) | 26,880 | (43) |
| Broadband | 394 | (9.4) | 9,869 | (19) | 275 | (7.1) | 13,054 | (17) | 375 | (7.9) | 8,513 | (14) |
| Dial-Up | 239 | (5.7) | 956 | (1.9) | 163 | (4.2) | 343 | (0.44) | 276 | (5.8) | 1,018 | (1.6) |
| IRC Server | 110 | (2.6) | 461 | (0.91) | 88 | (2.3) | 2,289 | (2.9) | 111 | (2.3) | 6,476 | (10) |
| Nameserver | 124 | (3.0) | 453 | (0.89) | 84 | (2.2) | 2,796 | (3.6) | 90 | (1.9) | 451 | (0.72) |
| Router | 58 | (1.4) | 2,698 | (5.3) | 76 | (2.0) | 4,055 | (5.2) | 125 | (2.6) | 682 | (1.1) |
| Web Server | 54 | (1.3) | 393 | (0.77) | 64 | (1.7) | 5,674 | (7.3) | 134 | (2.8) | 730 | (1.2) |
| Mail Server | 38 | (0.91) | 156 | (0.31) | 35 | (0.90) | 71 | (0.09) | 26 | (0.55) | 292 | (0.47) |
| Firewall | 9 | (0.22) | 7 | (0.01) | 3 | (0.08) | 3 | (0.00) | 2 | (0.04) | 1 | (0.00) |

Table 6: Breakdown of victim hostnames.

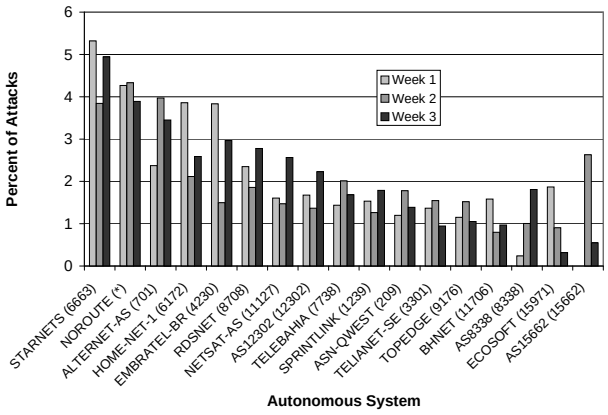



Figure 8: Distribution of attacks to Autonomous Systems (ASes) that were targeted by at least 1% of all attacks.

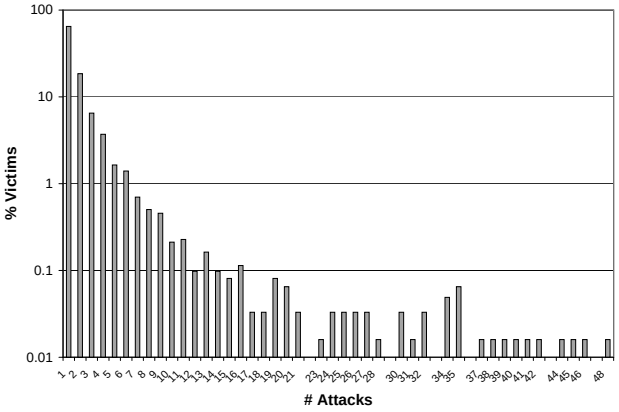



Figure 9: Histogram counting the number of victims of repeated attacks across all traces.

with the victim of an attack, we performed longest prefix matching against a BGP routing table using the victim's IP address. To construct this table, we took a snapshot from a border router with global routes on February 7, 2001. We then mapped AS numbers to identifying names using the NetGeo [17] service to do lookups in registry whois servers. We labeled addresses which had no matching prefix as "NOROUTE".

Figure 8 shows the distribution of attacks to the 17 ASes that were targeted by at least 1% of all attacks. As with top-level domains, each AS named on the x-axis is associated with three values, one for each of the three weeks of our study in progressive shades of grey. Note that the ASes are sorted by overall attacks across all three weeks.

From Figure 8, we see that no single AS or small set of ASes is the target of an overwhelming fraction of attacks: STARNETS was attacked the most, but only received 4-5% of attacks. However, the distribution of ASes attacked does have a long tail. The ASes shown in Figure 8 accounted for 35% of all attacks, yet these ASes correspond to only 3% of all ASes attacked. About 4% of attacks each week had no route according to our offline snapshot of global routes.

Compared with TLDs, ASes experienced more variation in the number of attacks targeted at them for each week. In other words, there is more stability in the type or country of victims than the ASes in which they reside. For example, EMBRATEL's percentage of attacks varies by more than a factor of 2, and AS 15662, an unnamed AS in Yugoslavia, did not show up in week 1 of the traces.

### 6.3.4 Victims of repeated attacks

Figure 9 shows a histogram of victims of repeated attacks for all traces combined. The values on the x-axis correspond to the number of attacks to the same victim in the trace period, and the values on the y-axis show what percentage of victims were attacked a given number of times in logarithmic scale. For example, the majority of victims (65%) were attacked only once, and many of the remaining victims (18%) were attacked twice. Overall,

most victims (95%) were attacked five or fewer times. For the remaining victims, most were attacked less than a dozen times, although a handful of hosts were attacked quite often. In the trace period, one host was attacked 48 times for durations between 72 seconds and 5 hours (at times simultaneously). The graph is also truncated: there are 5 outlier victims attacked 60–70 times, and one unfortunate victim attacked 102 times in a one week span.

### 6.4 Validation

The backscatter hypothesis states that unsolicited packets represent responses to spoofed attack traffic. This theory, which is at the core of our approach, is difficult to validate beyond all doubt. However, we can increase our confidence significantly through careful examination of the data and via related experiments.

First, an important observation from Table 3 is that roughly 80% of attacks and 98% of packets are attributed to backscatter that does not itself provoke a response (e.g. TCP RST, ICMP Host Unreachable). Consequently, these packets could not have been used for probing our monitored network; therefore network probing is not a good alternative explanation for this traffic.

Next, we were able to duplicate a portion of our analysis using data provided by Vern Paxson taken from several University-related networks in Northern California. This new dataset covers the same period, but only detects TCP backscatter with the SYN and ACK flags set. The address space monitored was also much smaller, consisting of three /16 networks ($\frac{3}{65536}$'s of the total IP address space). For 98% of the victim IP addresses recorded in this smaller dataset, we find a corresponding record at the same time in our larger dataset. We can think of no other mechanism other than backscatter that can explain such a close level of correspondence.

Finally, Asta Networks provided us with data describing denial-of-service attacks directly detected at monitors covering a large backbone network. While their approach and ours capture different sets of attacks (in part due ingress filtering as discussed in Section 3 and in part due to limited peering in the monitored network), their data qualitatively confirms our own; in particular we were able to match several attacks they directly observed with contemporaneous records in our backscatter database.

## 7 Related work

While denial-of-service has long been recognized as a problem [14, 18], there has been limited research on the topic. Most of the existing work can be roughly categorized as being focused on tolerance, diagnosis and localization. The first category is composed of both approaches for mitigating the impact of specific attacks [4, 16] and general system mechanisms [25, 1] for controlling resource usage on the victim machine. Usually such solutions involve a quick triage on data packets so minimal work is spent on the attacker's requests and the victim can tolerate more potent attacks before failing. These solutions, as embodied in operating systems, firewalls, switches and routers, represent the dominant current industrial solution for addressing denial-of-service attacks.

The second area of research, akin to traditional intrusion detection, is about techniques and algorithms for automatically detecting attacks as they occur [22, 13]. These techniques generally involve monitoring links incident to the victim and analyzing patterns in the arriving and departing traffic to determine if an attack has occurred.

The final category of work, focuses on identifying the source(s) of DoS attacks in the presence of IP spoofing. The best known and most widely deployed of these proposals is so-called *ingress* and *egress* filtering [12, 5]. These techniques, which differ mainly in whether they are manually or automatically configured, cause routers to drop packets with source addresses that are not used by the customer connected to the receiving interface. Given the practical difficulty of ensuring that all networks are filtered, other work has focused on developing tools and mechanisms for tracing flows of packets through the network independent of their ostensibly claimed source address [3, 26, 23, 2, 24, 11].

There is a dearth of research concerned with quantifying attacks within the Internet – denial-of-service or otherwise. Probably the best known prior work is Howard's PhD thesis – a longitudinal study of incident reports received by the Computer Emergency Response Team (CERT) from 1989 to 1995 [15]. Since then, CERT has started a new project, called AIR-CERT, to automate the collection of intrusion detection data from a number of different organizations, but unfortunately their results are not yet available [7]. To our knowledge ours is the only quantitative and empirical study of wide-area denial-of-service attacks to date.

## 8 Conclusions

In this paper we have presented a new technique, "backscatter analysis," for estimating denial-of-service attack activity in the Internet. Using this technique, we have observed widespread DoS attacks in the Internet, distributed among many different domains and ISPs. The size and length of the attacks we observe are heavy-tailed, with a small number of long attacks constituting a significant fraction of the overall attack volume. Moreover, we see a surprising number of attacks directed at

a few foreign countries, at home machines, and towards particular Internet services.

## Acknowledgments

We would like to thank a number of people for their contributions to this project. We are particularly grateful to Brian Kantor and Jim Madden of UCSD who provided access to key network resources and helped us understand the local network topology. kc claffy and Colleen Shannon at CAIDA provided support and valuable feedback throughout the project. David Wetherall and Gretta Bartels at Asta Networks donated their time, data and insight. Vern Paxson of ACIRI also provided valuable data and feedback at several stages of our thinking. Finally, we thank the anonymous reviewers for their comments and suggestions. Support for this work was provided by DARPA NGI Contract N66001-98-2-8922, NSF grant NCR-9711092, and Asta Networks.

## References

[1] Gaurav Banga, Peter Druschel, and Jeffrey Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In *Proceedings of the 1999 USENIX/ACM Symposium on Operating System Design and Implementation*, pages 45–58, February 1999.

[2] Steven M. Bellovin. ICMP Traceback Messages. Internet Draft: draft-bellovin-itrace-00.txt, March 2000.

[3] Hal Burch and Bill Cheswick. Tracing Anonymous Packets to Their Approximate Source. In *Proceedings of the 2000 USENIX LISA Conference*, pages 319–327, New Orleans, LA, December 2000.

[4] Cisco Systems. Configuring TCP Intercept (Prevent Denial-of-Service Attacks). Cisco IOS Documentation, December 1997.

[5] Cisco Systems. Unicast Reverse Path Forwarding. Cisco IOS Documentation, May 1999.

[6] Computer Emergency Response Team. CERT Advisory CA-1996-21 TCP SYN Flooding Attacks. `http://www.cert.org/advisories/CA-1996-21.html`, September 1996.

[7] Computer Emergency Response Team. AirCERT. `http://www.cert.org/kb/aircert/`, 2000.

[8] Computer Security Institute and Federal Bureau of Investigation. 2000 CSI/FBI Computer Crime and Security Survey. Computer Security Institute publication, March 2000.

[9] R. D'Agostino and M. Stephens. *Goodness-of-Fit Techniques*. Marcel Dekker, Inc., 1986.

[10] Tina Darmohray and Ross Oliver. Hot Spares For DoS Attacks. *;login:*, 25(7), July 2000.

[11] Drew Dean, Matt Franklin, and Adam Stubblefield. An Algebraic Approach to IP Traceback. In *Proceedings of the 2001 Network and Distributed System Security Symposium*, San Diego, CA, February 2001.

[12] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing. RFC 2827, May 2000.

[13] Mark Fullmer and Steve Romig. The OSU Flow-tools Package and Cisco Netflow logs. In *Proceedings of the 2000 USENIX LISA Conference*, New Orleans, LA, December 2000.

[14] Virgil Gilgor. A Note on the Denial-of-Service Problem. In *Proceedings of the 1983 IEEE Symposium on Security and Privacy*, Oakland, CA, 1983.

[15] John D. Howard. *An Analysis of Security Incidents on the Internet*. PhD thesis, Carnegie Mellon University, August 1998.

[16] Phil Karn and William Simpson. Photuris: Session-Key Management Protocol. RFC 2522, March 1999.

[17] David Moore, Ram Periakaruppan, Jim Donohoe, and kc claffy. Where in the World is netgeo.caida.org? In *INET 2000 Proceedings*, June 2000.

[18] Roger Needham. Denial of Service: An Example. *Communications of the ACM*, 37(11):42–47, November 1994.

[19] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis. RFC 2330: Framework for IP performance metrics, May 1998.

[20] Vern Paxson. Personal Communication, January 2001.

[21] Jon Postel, Editor. Internet Control Message Protocol. RFC 792, September 1981.

[22] Steve Romig and Suresh Ramachandran. Cisco Flow Logs and Intrusion Detection at the Ohio State university. *login; magazine*, pages 23–26, September 1999.

[23] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical Network Support for IP Traceback. In *Proceedings of the 2000 ACM SIGCOMM Conference*, pages 295–306, Stockholm, Sweden, August 2000.

[24] Dawn Song and Adrian Perrig. Advanced and Authenticated Marking Schemes for IP Traceback. In *Proceedings of the 2001 IEEE INFOCOM Conference*, Anchorage, AK, April 2001.

[25] Oliver Spatscheck and Larry Peterson. Defending Against Denial of Service Attacks in Scout. In *Proceedings of the 1999 USENIX/ACM Symposium on Operating System Design and Implementation*, pages 59–72, February 1999.

[26] Robert Stone. CenterTrack: An IP Overlay Network for Tracking DoS Floods. In *Proceedings of the 2000 USENIX Security Symposium*, pages 199–212, Denver, CO, July 2000.