A Zero-Pass End-to-End Checksum Mechanism for IPv6[1]

(CREATION DATE: January 13, 1995)
(VERSION DATE: December 1, 1995 3:37 pm)

Gregory G. Finn, Steven Hotz, Craig Milo Rogers
USC/Information Sciences Institute

1. Introduction

A primary source of overhead in protocol processing can be attributed to operations that read and/or write all, or most, of the bytes in a packet. Examples of these operations are digest or encryption security, buffering and copying between different protocol layers (network -vs- transport) or implementation layers (application -vs- operating system), or data fidelity checksum algorithms.

As local area networks (LANs) approach gigabit channel speeds, the pressure to reduce protocol overhead has risen. A great deal of research and development effort has been expended to improve transport-layer performance. One focus of that research has been mechanisms that speed up end-to-end checksum insertion and validation.

TCP and UDP are examples of protocols for which all the protocol-processing information is placed in the header of a packet. In particular, the end-to-end checksum in both TCP and UDP is placed in the headers of their packets rather than in trailers. It has been argued that locating the end-to-end checksum in the header is a source of unneccessary overhead, since it prevents source hardware from performing 'on-the-fly' checksum calculation [1] [2][3][4][5].

We present a zero-pass mechanism for the creation and insertion of embedded end-to-end checksums that requires neither trailers nor specialized hardware assistance to transmit a packet. Source and destination hosts are not required to be on the same LAN and no changes are required to the proposed IPv6 protocol for it to operate.

[1] USC/ISI Patent Pending

2. Background

The pressure to increase the performance of distributed applications across the internetwork has been strongly felt for some time. In particular, there is a strong need to minimize the latency of the remote procedure call (RPC). Research and development efforts have focused on improving protocol software efficiency, the streamlining of protocol definitions, decreasing unneeded packet handling, adding specialized hardware to accelerate aspects of protocol processing and parallelizing of protocol processing.

The computation of embedded end-to-end checksums is a principal source of overhead when sending packets across an internetwork. Several techniques have been developed to minimize its cost or to reduce that cost to the point where it no longer limits bandwidth. However, little has been done to reduce transmission latency associated with its calculation.

The great majority of protocol stacks copy a packet at least once between the network interface and application memory. For RISC processors those copies are usually performed by programmed I/O loops. Although the copy operation greatly decreases performance, the end-to-end checksum computation can be combined with the copy so that it adds little overhead. Because memory access often requires at least two machine cycles, the accumulation of a checksum need not add overhead above that required for the copy. This was pointed out in [6].

Higher-performance can be achieved by copying the packet using hardware rather than software. To avoid reintroducing the need to scan the packet to determine the checksum, these network interfaces usually associate checksum calculation with the DMA transfer logic. Checksum calculation is then performed as a side-effect of transferring packets to and from the network interface across the system bus [7][8][3]. Parallelization of network-layer and transport-layer protocol processing has also been examined as a means of improving

This research was sponsored by the Advanced Research Projects Agency under Contract No. DABT63-93-C-0062. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policies, either expressed or implied, of ARPA, the U.S. Government, or any person or agency connected with them.

USC/Information Sciences Institute
4676 Admiralty Way
Marina Del Rey, CA 90202
Internet: Finn@isi.edu, Hotz@isi.edu, Rogers@isi.edu

performance [9]. Specialized logic to compute checksums is often incorporated into such designs.

To achieve highest performance, protocol stacks avoid packet copying altogether. In this situation applications share memory with the network interface. Here a zero-pass mechanism is needed for calculation and validation of end-to-end checksums, since there is no opportunity to scan the packet as it is copied.

Computing an end-to-end checksum can be entirely avoided by relying upon link-layer data protection as a substitute when it is available and the interface is judged to be sufficiently reliable [10]. However, this provides protection only when both the source and destination are on the same network. This sharply restricts the domain across which this technique is usable and encourages the development of applications that exhibit de facto internet incompatibility.

3. Zero-Pass Mechanism for Checksuming Network-Layer Payloads

Each incoming packet passes through the network interface input-channel logic before it enters system or buffer memory. This provides an opportunity to calculate an end-to-end checksum across the link-layer packet without requiring an extra pass over the protected data [3]. However, to calculate the correct end-to-end checksum as a packet streams in from the input channel, either the checksum must be gathered over only that data within the packet which is protected or subsequently, the checksum algorithm must allow adjustment to exclude data that should not participate.

The Internet checksum algorithm allows that adjustment [11][12][13][14]. Unwanted data can be removed simply by subtracting its contribution from the checksum value. The Internet checksum is also extremely simple to compute in both hardware and software. A 16-bit ones complement adder in the data path can accumulate the checksum at gigabit link speed using conventional CMOS logic.

If the link-layer does not fragment network-layer packets, the end-to-end checksum can be determined on input without the need for an additional scan across payloads. If payloads are fragmented, computing the checksum is considerably more complex.

3.1 Insertion of End-to-End Checksum

When any source on a network transmits a packet, that packet will be received by another interface on the same network. That first-hop receiver may be the packet's destination or a gateway. If the data protection provided by the link-layer is equal to or greater than that provided by the end-to-end checksum and the interfaces are sufficiently reliable, the link layer can be relied upon to provide end-to-end data fidelity for at least one hop. There is little need until the packet leaves the protection provided by the link layer for the end-to-end checksum to have been calculated or installed.

The protection provided by the link-layer extends the region within which the source can rely upon data fidelity beyond the source system for at least one hop. In general, loss of link-layer protection occurs when the packet is received at the first-hop, as it crosses the input interface and is stored. The end-to-end checksum should be calculated as it crosses that boundary and then be inserted into the packet.

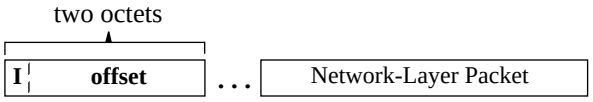
It is common for the first-hop away from the source to be either the destination or a gateway host. The first hop could also be a bridge or a switch. Whether or not the link-layer protection relied upon to postpone end-to-end checksum calculation and insertion extends beyond one hop depends upon the particulars of the source's network. In a switch-based network where the switches are of cut-through design, protection may extend for several hops, until the packet is received by the first host.[2]

By allowing a source to transmit network-layer packets onto its network without an end-to-end checksum, receivers in that network must distinguish two disjoint classes of packets. If a checksum has not been inserted, the receiver must calculate and insert the checksum. Otherwise the receiver may gather the checksum to validate the packet.

For example, assume that a field {**I**, **offset**} is associated with each network-layer packet that crosses a LAN. The flag **I** indicates whether or not

2. Since ATM fragments its payloads at the source and does not necessarily reassemble them after one hop, the first-hop receiver on an ATM-based network may not in general be able to insert a checksum.

a checksum has been inserted, while **offset** points to the end-to-end checksum field. Before the source transmits a packet it inserts the correct values for **offset** and sets **I** to indicate that the end-to-end checksum has not been inserted and transmits the packet without calculating the end-to-end checksum value.



Example: Link-Layer Flag and Offset

When a network-layer packet is received, the end-to-end checksum is calculated by the interface and subsequently, it is adjusted. If the value of **I** indicates that the end-to-end checksum has not been inserted, the checksum is correctly inserted into the packet at the location indicated by **offset** and **I** is set to indicate that the checksum is present in the packet. Subsequently, the packet contains the correct end-to-end checksum.

Otherwise, when a network-layer packet is received **I** indicates that the end-to-end checksum is present. No action is taken unless the receiving interface is the network-layer destination. In that case, the checksum value calculated on input is adjusted and used to validate the incoming packet.

3.2 Rules for Connected Gateways

The mechanism just described is now extended to packets that either enter or leave via gateways. Assume that all interfaces on a connected network or sub-net implement the mechanism, either with hardware assistance or in software. Consider network-layer packets that enter or leave a gateway. There are three cases that concern us:

1. The gateway forwards a packet from a connected network that does implement zero-pass into one that does not implement it.
2. The gateway forwards a packet from a connected network that does not implement zero-pass into one that does implement it.
3. The gateway forwards a packet between connected networks that both implement zero-pass.checksum.

In case 1, the end-to-end checksum is inserted by the gateway if it is the first-hop receiver. The gateway must forward the network-layer packet without the {**I**, **offset**} field. If the network-layer packet source is the gateway itself, the gateway must compute the checksum and insert it.

In case 2, the receiving gateway should add the {**I**, **offset**} field with **I** indicating that end-to-end checksum has been inserted. The **offset** should either be correct or indicate that the destination should parse the packet to find the checksum field location.

If the packet source is the gateway itself, **I** must indicate that the checksum has not been inserted and **offset** must be correct.

For case 3, no special action is necessary.

3.3 Extension to Bridging

The zero-pass mechanism as described can be applied to bridged LANs. In the case of a packet traveling from the side that does implement zero-pass to another side, the situation is similar to forwarding by a gateway. There are three cases that concern us:

1. The bridge forwards a packet from a connected network that does implement zero-pass into one that does not implement it.
2. The bridge forwards a packet from a connected network that does not implement zero-pass into one that does implement it.
3. The bridge forwards a packet between connected networks that both implement zero-pass.

In case 1, the end-to-end checksum is inserted by the bridge if it is the first-hop receiver. The {**I**, **offset**} field is stripped by the link-layer and the resulting packet is forwarded to the other side.

In case 2, the receiving bridge should add the {**I**, **offset**} field with **I** indicating that end-to-end checksum has been inserted. The **offset** should either be correct or indicate that the destination should parse the packet to find the checksum field location.

If the packet source is the bridge itself, **I** must indicate that the checksum has not been inserted and **offset** must be correct.

In case 3, no special action need be taken.

Notice that when a packet is forwarded from a network or sub-net that does not implement the zero-pass mechanism into one which does implement it, the **offset** value may require that the destination parse its headers to find the checksum field location.

3.4 Pros and Cons

The zero-pass mechanism postpones calculation and insertion of an embedded checksum for transmission by one hop. This transforms that

checksum calculation from scan-before-output into a scan-on-input. This transformation eliminates the performance penalty associated with an embedded checksum and provides several cost and performance advantages:

1. Checksum hardware assistance is used only in the input channel. No checksum hardware assistance is needed for output.
2. Since checksum calculation is associated with the channel logic, there is no need to copy a packet to perform the checksum calculation.
3. Transmission delay is minimal. No checksum calculation is performed before transmission.
4. In general, the costs for validation and calculation-insertion of the checksum are insensitive to packet size.

3.4.1 Criticisms

Zero-pass requires that a source trust the first-hop receiver to correctly calculate and insert the end-to-end checksum, whereas previously, the source performed those operations itself. This raises an issue of network interface compatibility within the source's network. There is no requirement imposed outside the source's network.

It is not necessary that all interfaces on a LAN or subnet be required to implement the zero-pass mechanism for it to be used. While it is most effective if that is done, for existing LANs it is possible to define a new packet type that encapsulates the {**I**, **offset**} and network-layer packet. This new packet would be exchanged across the LAN between interfaces that implement zero-pass. Otherwise, non-encapsulated packets for which zero-pass is not used would be exchanged. Network interfaces that do not recognize the new packet type would either discard it or complain to the source. This provides a means by which an interface could determine whether or not another interface on the LAN implements zero-pass.

Zero-pass does require that the first-hop receiver insert the end-to-end checksum when one is not already present. That cost need be no larger than one test plus several instructions per packet. Network-layer packets remain unaltered outside the LAN.

Criticisms of relying exclusively upon link-layer protection were made in [6]:

1. That subnetting made it difficult to determine from the destination address whether or not a packet will stay within a network.
2. If the checksum comes freely as part of a copy loop, why remove it from that loop?
3. LAN interfaces may corrupt their packets in ways not detected and reported to their hosts.

Since the zero-pass operates correctly across both bridges and gateways, a source need not determine whether or not the destination address is within its LAN or subnet.

The second objection concerns low-performance systems, which use programmed I/O loops to copy packets to and from the network interface. Zero-pass is not expected to improve their performance. For high-performance interfaces, removing the need to calculate an end-to-end checksum before transmission removes the need to have checksum logic in the transmit-side of the interface. It also reduces transmission latency by the time that would otherwise be needed to scan the packet to calculate the checksum.

An implementation of zero-pass should compute the end-to-end checksum as each packet is received over the channel, rather than subsequently, by a scan across the packet in interface memory. Aside from the obvious performance objection, this protects the network-layer packet from possible memory corruption within a first-hop receiver.

However, there remains the possibility that a packet may become corrupted in memory prior to its transmission. Although this problem can be addressed by standard techniques that discover and report memory corruption, many computer systems no longer provide mechanisms to detect data corruption within their memory nor do they protect data when moved across a bus.

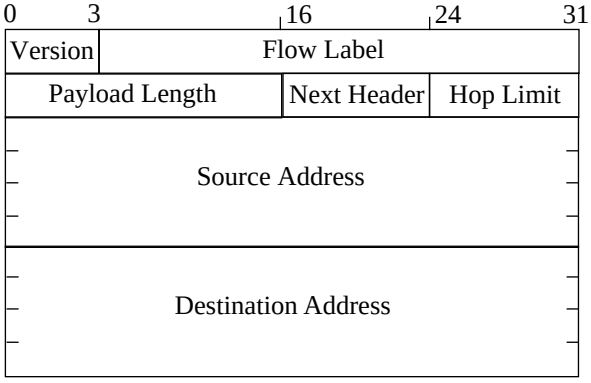
4. Adjustment Cost for IPv6

Assume that an interface implements the zero-pass mechanism. What is the cost to adjust the checksum that was computed by the interface across an IPv6 packet [18] so that it represents the correct transport-layer checksum value? Ideally, the interface should gather a checksum across only those octets that participate in the transport-layer checksum. But it is unreasonable to require that hardware parse network-layer and transport-layer packets.

If the checksum has been gathered across the entire link-layer packet, the contribution made by the link-layer header/trailer octets must be removed. A more effcient alternative would have the link layer

providing assistance in its header that describes where the embedded IP packet starts and ends. This could be used to direct the checksum hardware to compute the checksum across only those octets of the enclosed IP packet.

A checksum calculated across the entire IP packet must still be adjusted to be the correct transport-layer checksum for UDP or TCP payloads. Assume that the destination host machine supports an idealized set of RISC machine instructions that includes 16-bit ones complement arithmetic, byte and word loads and stores, and 8-bit logical shifts. To avoid generation of +0, subtraction assumes one's complement followed by addition.



Proposed IPv6 Header

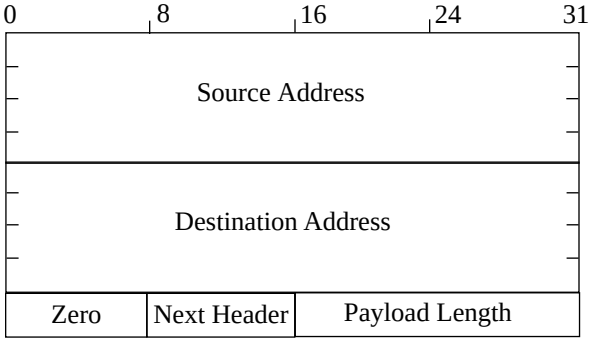
4.1 Checksum Across Transport Layer

Since all the transport-layer octets participate in the checksum, the cost of adjustment focuses on transforming the checksum so that it covers the correct pseudo header.

4.1.1 Without Extension Headers

The most commonly expected case is to receive an IPv6 packet that carries a UDP or TCP payload and that does not contain an extension header. To adjust the checksum when there are no extension headers requires that the contributions of the Version, Flow Label and Hop Limit be removed and that the contribution of the Next Header field be shifted to the least significant byte of the checksum.

Removing the Version and Flow Label from the checksum requires two loads and two ones complement subtractions. Removing the Next Header and Hop Limit fields requires one load and one subtraction. Logical shifting the Next Header field right eight bits and adding it back requires two more operations



IPv6 Pseudo Header

```
LOAD    CHK,  <checksum>
LOAD     R1,  <Version, high Flow Label>
SUB     CHK,  R1
LOAD     R1,  <low Flow Label>
SUB     CHK,  R1
LOAD     R1,  <Next Header, Hop Limit>
SUB     CHK,  R1
LSH      R1,  #8          ; right shift
ADD     CHK,  R1
```

Although it is depicted in a different spot in the diagram of the pseudo header in [18], the Payload Length contribution to the checksum is unaffected, since it remains 16-bit aligned. The typical cost to correctly adjust the computed checksum value when a packet does not contain extension headers is 9 instructions. Determining whether or not an extension header is present would add another two instructions for an indexed branch.

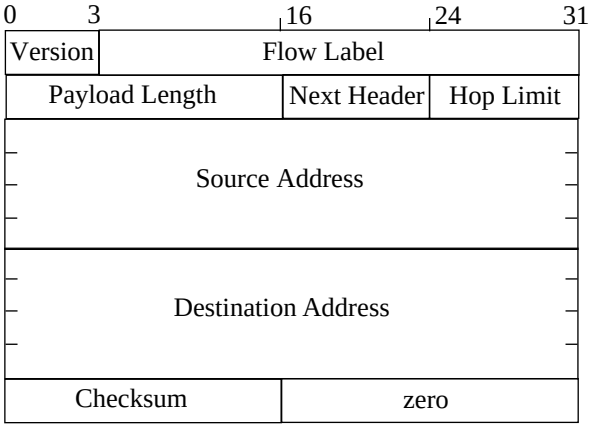
4.1.2 With Extension Headers

Less commonly expected is an IPv6 packet that carries an extension header. The cost of adjusting the checksum value rises markedly when extension headers are present. The extension headers must be parsed and the contribution of each of the extension header words must be removed from the checksum value.

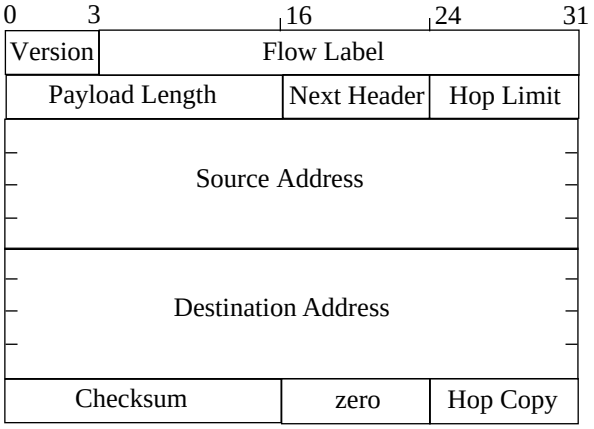
4.2 Checksum Across Network-Layer

We now redefine IPv6 so that the costs of adjusting the checksum at the first-hop receiver are reduced. Assume that the entire IPv6 packet is covered by a checksum. Although a checksum is calculated across the header, intermediate nodes need not recalculate the checksum. However, as a consequence a mechanism must exist that allows the checksum to remain valid when intermediate nodes modify the header.

There is no longer need for a separate transport-layer checksum. The UDP and TCP checksum fields could remain for purposes of compatibility



New IPv6 Header



New IPv6 Header with Hop Copy

with IPv4 and to ease v4 to v6 conversion. The transport-layer would not need to calculate a checksum.

The new IPv6 header would contain two additional fields, Checksum and two unused octets. The Checksum field contains the 16-bit Internet checksum across the entire IPv6 packet. Having the Checksum field at a known location facilitates adjustment of the checksum by intermediary nodes to maintain the validity of the checksum.

The header now contributes to the checksum. Changes made en route to the header must be reflected in changes to the checksum field in the IPv6 packet. In the common case, routers need to add one to the checksum field when the Hop Limit is decremented. This costs a router five instructions using the idealized 16-bit machine.

```
LOAD    CHK,  <checksum value in packet>
NOT     CHK,  CHK
ADD     CHK,  #1
NOT     CHK,  CHK
STORE   CHK,  <checksum value in packet>
```

When received at the destination node, there would be no need to further adjust the checksum once the link-layer contribution was removed.

The requirement that routers adjust the header can be eliminated by adding a Hop Copy field to the IPv6 header. The Hop Copy field would be loaded with the same value as the Hop Limit field by the originating node. At a destination, the difference beteeen the Hop Copy and Hop Limit fields would be added back into the checksum. This costs seven instructions.

```
LOAD    CHK,  <checksum>
NOT     CHK,  CHK
LOADB     R1,  <Hop Limit>
LOADB     R2,  <Hop Copy>
SUB       R2,  R1
ADD     CHK,  R2
NOT     CHK,  CHK
```

Because the extension header contributes to the checksum, at the destination there is no cost of removing its contribution. Intermediate nodes that modify an extension header must make a complementay change to the Checksum field. Currently, this would be required only for the Loose Source Route extension header.

4.2.1 Summary of Adjustment Cost

Under the currently proposed definition of IPv6, the cost to adjust an Internet checksum that is been accumulated by hardware across an unfragmented IPv6 packet, is approximately 11 instructions when the IPv6 packet contains no extension headers. When the packet does contain an extension header, an additional cost accrues that is proportional to the length of the extension header.

IPv6 can be redefined to support a network-layer checksum across the entire IPv6 packet, while requiring no additional work by intermediate routers. If that is done, the cost to adjust the checksum at the destination is fixed at approximately seven instructions. If intermediate routers make a complementary addition to the checksum to account for decrementing the Hop Limit field, no adjustment of the checksum would be needed at the destination.

5. Conclusion

A zero-pass mechanism for calculation and validation of embedded end-to-end checksums has been defined. In those networks that provide link-level data error detection, a source need not calculate an end-to-end checksum before packet

transmission. This leads to a significant decrease in RPC latency. The zero-pass mechanism has no effect upon network-layer or transport-layer packet format outside of LANs that use it.

The advantage of zero-pass is most evident when the source-to-destination propagation time is small. By eliminating the need to scan packet to calculate a checksum payload data prior to transmission, transport-layer packets may be transmitted directly from application memory onto the network.

It is hoped that use of zero-pass will discourage the use of specialized intra-network transport protocols or the use of non-checksummed data transmission, since a reliance upon those in applications can lead to internet incompatibility.

6. Acknowledgments

The authors thank Joe Touch of USC/ISI for some helpful discussions.

7. References

[1] Weaver, A.
Making Transport Protocols Fast
Proceedings of 16th Conference on Local Computer Networks, pp. 295-300,
IEEE Press, 1991.

[2] Whaley, A.
The XpressTransfer Protocol
Proceedings of 14th Conference on Local Computer Networks, pp. 399-407,
IEEE Press, 1989.

[3] Steenkiste, P.
A Systematic Approach to Host Interface Design for High-Speed Networks
IEEE Computer, March 1994.

[4] Williamson, C., Cheriton, D.
An Overview of the VMTP Transport Protocol
Proceedings of 14th Conference on Local Computer Networks, pp. 399-407,
IEEE Press, 1989.

[5] Bridges, M., Subramaniam, S., Edwards, D.
Internet Draft
TCP Embedded Trailer Checksum
Hewlett-Packard, December 1994.

[6] Partridge, C.
Gigabit Networking (Ch. 10)
Addison-Wesley,1994.

[7] Boden, N., Cohen, D., et al.
Myrinet: A Gigabit-per-Second Local Area Network
IEEE Micro, Vol. 15, No. 1, February 1995.

[8] Siegel, M., Williams, M., Roßler, G.
Overcoming Bottlenecks in High-Speed Transport Systems
Proceedings of 16th Conference on Local Computer Networks, pp. 399-407,
IEEE Press, 1991.

[9] Koufopavlou, O., Tantawy, A., Zittrbart, M.
Analysis of TCP/IP for High Performance Parallel Implementations
Proceedings of 17th Conference on Local Computer Networks, pp. 576-585,
IEEE Press, 1992.

[10] Cheriton, D.
RFC 1045: VMTP: Versatile Message Transaction Protocol
Stanford University, February 1988.

[11] Plummer, W
IEN 45: TCP Checksum Function Design
Bolt Beranek and Newman, Inc., June 1978

[12] Braden, R., Borman, D., Partridge, C.
RFC 1071: Computing the Internet Checksum
September 1988.

[13] Mallory, T., Kullberg, A.
RFC 1141: Incremental Updating of the Internet Checksum
BBN Communications, January 1990.

[14] Rijsinghani, A. (editor)
RFC 1624: Computation of the Internet Checksum via Incremental Update
Digital Equipment Corp., May 1994.

[15] Postel, J. (editor)
RFC 791: Internet Protocol

[16] Postel, J.
RFC 768: User Datagram Protocol

[17] Postel, J. (editor)
RFC 793: Transmission Control Protocol

[18] Editor: Hinden, R
Internet Protocol, Version 6 Specification (currently in draft form).

[19] Bradner, S., Mankin, A..
RFC 1752: The Recommendation for the Next Generation Protocol

[20] R*FC 1726*: Technical Criteria for Choosing IP The Next Generation (IPng)
Partridge, C., Kastenholz, F.
FTP Software, December 1994.