

1. AIM

This working aide will attempt to introduce the analysts to the basic concepts required to understand network congestion and congestion control, provide an introduction to ECN (Explicit Congestion Notification) and demonstrate basic TCP flag analysis methodology.

The development of this working aide was necessitated by recent occurrences when junior analysts misinterpreted ECN traffic observed on the network, clearly indicating that:

- a. the concept of ECN, and congestion avoidance/control in general, is a foreign concept to junior analysts;
- b. junior analysts are either unfamiliar with or require a basic refresher of TCP flag analysis; and
- c. the junior analysts are either unfamiliar with or require a basic refresher in the construction of TCPDump filters and bitmasks.

The fact that the traffic in question was misinterpreted in this case was quite understandable, as most analysts are not introduced to the concept of congestion control and the TCP/IP ECN implementation until they receive their SANS GIAC training.

Although TCP flag analysis and bitmasking are introduced during initial contact training and will be covered in depth during their SANS track, it is clear that additional guidance is required for the interim.

2. CONGESTION CONTROL - AN INTRODUCTION

2.1 Rise of the Beast - Network Congestion Manifests

In the early to mid 1980s, back when the World Wide Web was but a glimmer in Tim Berners-Lee's¹ eye, computer networks experienced a period of rapid expansion as the seeds of what would evolve into our modern Internet were sown.

Unfortunately, this period of explosive expansion resulted in the genesis of a new problem that began to plague the NSFNet²- network congestion. At first, the congestion problem was manageable, resulting in the dropping of a mere 10-20% of packets due to queue buffer overflows.

Suddenly, in October 1986, this problem came to a head as the internet experienced the first of many "congestion collapses" during which network throughput was reduced by a "factor of thousands" drop in bandwidth. During this collapse, UC Berkeley/LBL saw the throughput of one of their networks drop from 32 Kbps to less than 40 bps.

¹ One of the inventors of the World Wide Web and president of the World Wide Web Consortium (W3C).

² A wide area network established by the National Science Foundation that replaced ARPANet.

The above incident allows one to fully appreciate the consequences of uncontrolled network congestion, especially when one considers that the internet link between UC Berkeley and LBL endpoints were separated by a mere 365 metres and two IMP³ hops.

Shortly after the Berkeley/LBL incident, researchers at Berkeley began to investigate the situation, analyzing the 4.3 BSD⁴ TCP implementation in an effort to slay the beast; this investigation revealed that although there were problems with the 4.3 implementation, the problems were addressable (pun intended).

2.2 Post Proelia Praemia⁵ - TCP Redux

Several kludges⁶ were developed for the 4.3 release as the result of this research and new features were introduced in order to address the burgeoning network congestion problems. Eventually 4.3 kludges were refined and fully implemented, along with other improvements, into 4.4BSD TCP.

Some of the more significant of these that are still utilized in modern TCP implementations are:

- a. **slow start** - Used at the beginning of a transfer or after repairing detected packet loss slowly probe the network to determine its available capacity;
- b. **fast retransmit** - Used by the TCP sender to detect and repair loss based upon incoming duplicate ACKs. After the arrival of 3 duplicate ACKs, TCP performs a retransmission of what appears to be the missing segment⁷ without waiting for the retransmission timer to expire;
- c. **aggressive TCP receiver ACK policy** - The most significant improvements under this policy are:
 - i. an ACK should be generated for at least every second data segment and must be generated within 500ms of the arrival of the first unACKed packet;
 - ii. out-of-order data segments should be acknowledged immediately in order to accelerate loss recovery; and
 - iii. A TCP receiver must not generate more than one ACK for every incoming segment, other than to update the offered window size;

³ IMP (Interface Message Processor) - device used to connect to the ARPANet and the original NSFNet - the IMP was a precursor to today's routers.

⁴ Berkeley Software Distribution UNIX - 4.3 BSD was released in June 1986; the version implementing the recommendations in Jacobson & Karels' 1988 paper "Congestion Avoidance and Control" was designated "4.3 BSD Tahoe".

⁵ Latin - "After the battle comes the rewards".

⁶ Kludge (pronounced "klooj") - in computing, an elegant patchwork solution that is hastily implemented in order to address problems in a software or hardware implementation.

⁷ The terms "segment/TCP segment" and "packet/TCP packet" are often interchangeable within RFCs.

- d. **dynamic window sizing on congestion** - Combats congestion through the dynamic alteration of the TCP's sliding window size. The algorithm follows three simple rules:
 - i. if the RTO expires, set the congestion window size to half of its current size;
 - ii. increase the congestion window size by 1 segment for each acknowledgement; and
 - iii. use the minimum of either the receiver's advertised window or the congestion window;
- e. **round trip time variance estimation** - The TCP RTO (Retransmission Time Out) estimator was improved by the addition of an algorithm that takes into consideration possible variances in round trip times - the new estimator is capable of adapting to much greater traffic loads; and
- f. **Karn's Exponential RTO Back-off Algorithm** - Older TCP implementations would worsen network congestion due to multiple, successive retransmits as the RTO estimator was only updated upon receipt of an acknowledgement - if data is continually lost due to high network loading, the data would be retransmitted at constant intervals. Phil Karn developed a strategy for the AX.25⁸ a strategy that handles multiple retransmits with an exponential back-off of the RTO timer for each successive retransmit.

Although Today's TCP implementations utilize several other improvements to increase network throughput and prevent network congestion, the improvements suggested by Jacobson & Karels set the stage for the development of the TCP implementation we know and love today.

⁸ The AX.25 specification is Amateur Radio's implementation of the X.25 data link layer protocol suite. X.25 is an ITU-T standard protocol suite for WAN networks using the phone or ISDN system as the networking hardware; it defines the first three layers of the OSI model, though in the case of AX.25, the network layer is seldom used.

3. APPLICABLE RFCs⁹

3.1 RFC 1349 - Type of Service in the Internet Protocol Suite

RFC 1349, issued in July 1992, defined the most recent I implementation of the ToS field in the IP header. RFC 3168 superseded the original ToS field designation; this RFC redefined the ToS field in order to facilitate the implementation of an ECN capability in IP.

3.2 RFC 2474 - Definition of DS Field in the IPv4 and IPv6 Headers

RFC 2474, issued in December 1998, details the implementation of the differentiated services field within IPv4 and IPv6 headers; bits 7 and 8 of the DS field is currently employed to implement ECN in IP, though this RFC shows them as "unused".

3.3 RFC 2481 - A Proposal to add Explicit Congestion Notification to IP

RFC 2481, issued in January 1999, issued the first proposal to add explicit congestion notification to IP; RFC 3168 has since superseded it.

3.4 RFC 2581 - TCP Congestion Control

RFC 2581, issued in April 1999, details TCP's built-in congestion control algorithms and describes the concepts of the congestion window and maximum segment sizes.

3.5 RFC 2780 - Allocation Guidelines For Values in IP and Related Headers

RFC 2780, issued in March 2000, details the currently accepted IANA guidelines for the allocation of specific values within the IP header; this RFC designates bytes 7 and 8 of the DS field as being allocated to ECN values.

3.6 RFC 3168 - Addition of Explicit Congestion Notification (ECN) to IP

RFC 3168, issued in September 2001, detailed the addition of two additional flags to the TCP header by the utilization of two bits of the reserved field.

⁹ RFC – “Request for Comments”. RFCs are a series of technical notes about the Internet first implemented in 1969 in order to propose new Internet standards. An RFC can be submitted to the Internet Engineering Task Force by anyone; eventually, if it gains enough interest, it may evolve into a genuine Internet standard. Each RFC is designated by an RFC number; once published, an RFC never changes - modifications to an original RFC are assigned a new RFC number.

4. NETWORK CONGESTION CONTROL CONCEPTS & DEFINITIONS

4.1 Quality of Service (QoS)

On a packet switched network such as the internet, quality of service refers to the probability of a packet succeeding in passing between two points in the network.

Network QoS can be improved by the use of queue buffers on routing devices and the implementation of various congestion control mechanisms

4.2 Congestion Avoidance/Congestion Control

Congestion avoidance techniques monitor network traffic loads in an effort to anticipate and avoid network congestion and maintain QoS; this is most often achieved via the dropping of packets.

Congestion Control refer to a means by which the circulation of network traffic flows are controlled in order to avoid its network congestion and the accompanying degradation of QoS.

4.3 Queue Buffer

Implementation of queue buffers on routing devices allows a significant improvement in the quality of service (QoS) on a network; a queue buffer is a means for storing a number of information signals without priority in a sending sequence.

In the case of a routing device, this consists of dedicated memory that either stores a packet that is awaiting transmission over a network or a packet that has been received over a network.

4.4 Queue Length

The queue length of a particular traffic flow is determined by calculating the number of packets in the flow; the number of packets in the queue in turn determines the amount of packet buffer space that is allocated to a given flow.

When a flow has a high queue length, the computed value is lowered, thereby allowing new incoming flows to receive free buffer space in the packet queue; when implemented, this technique allows all flows to get a proportional share of packets through the queue.

4.5 Congestion Window

Congestion window is TCP state variable that limits the amount of data a TCP connection can send; the size of the congestion window is increased or decreased depending on network congestion levels.

The implementation of congestion control mechanisms allows a routing device to alter its congestion window size "on the fly" as congestion conditions fluctuate.

Changing the window size advertised by a receiving endpoint to a peer on a network connection allows the device to control the rate at which the transmitting endpoint transmits data; this is the means by which the sliding TCP receive window size system implements flow control between two connected endpoints.

The TCP receive window size is the amount of received data (in bytes) that can be buffered during a connection; the transmitting endpoint can send only that amount of data before it must wait for an acknowledgment and window size update from the receiving endpoint.

4.6 Active Queue Management (AQM)

The implementation of ECN tags within the TCP and IP headers allow for the implementation of an AQM mechanism within both IPv4 and IPv6.

AQM is a proactive QoS (Quality of Service) method of indicating network congestion before the packet queue buffer of a routing device overflows and begins dropping packets; AQM is accomplished utilizing technique called Dynamic Buffer Limiting.

4.7 Dynamic Buffer Limiting (DBL)

DBL functions by tracking the queue length for each traffic flow on a routing device; when the queue length of a flow exceeds its limit, DBL will either drop packets or set the ECN/CWR bits in the TCP header in an attempt to prevent network congestion and maintain QoS.

5. TCP/IP CONGESTION AVOIDANCE

5.1 Network Congestion

Network congestion problems may occur when many concurrent TCP connections are experiencing queue buffer drops; when this occurs, TCP's automatic congestion avoidance (detailed in RFC 2581) is insufficient. When this occurs, network QoS suffers.

5.2 Congestion Avoidance Techniques

Several congestion avoidance methodologies have been developed for packet switched networks; this include, but are not limited to:

- a. Tail Drop (TD);
- b. Random Early Detection (RED);
- c. Weighted Random Early Detection (WRED); and
- d. Explicit Congestion Notification (ECN).

5.2.1 Tail Drop

This is the default congestion avoidance method normally utilized by routing devices when another (e.g. WRED, RED - discussed below) is not configured; tail drop treats all traffic equally and does not differentiate between varying priorities.

When TD is implemented on a routing device, the device's buffer queue will begin to fill during periods of congestion. Once the queue has been filled and, packets are simply dropped until the congestion is eliminated and the queue is no longer full.

5.2.2 Random Early Detection (RED)

One solution to network congestion is to utilize a congestion avoidance mechanism designated RED (Random Early Detection) on the queue buffer. RED is a congestion avoidance mechanism that takes advantage of TCP's congestion control mechanism.

RED randomly drops packets prior to periods of high congestion, thereby signalling the packet source to decrease its transmission rate. Assuming the packet source is using TCP, it will decrease the rate of packet transmission until all the packets reach the destination, thereby indicating that the congestion is cleared.

5.2.3 Weighted Random Early Detection (WRED)

WRED differs from RED in that it provides separate thresholds and weights for different IP precedences, thereby allowing the routing device to prioritize traffic streams; standard traffic may be dropped more frequently than priority traffic during periods of congestion.

5.2.4 Explicit Congestion Notification (ECN)

ECN is only used when two hosts ECN capable hosts signal that its mutual use is desired; the use of ECN without proper negotiation could cause non-ECN capable network equipment to drop traffic in which an ECN bit has been set.

With this method, an ECN bit is used to signal that there is explicit congestion. This method is superior in several instances than the indirect packet congestion notification performed by the RED/WRED algorithms; however, ECN requires explicit support by both hosts to be effective.

When an ECN capable routing device receives packet marked as ECN capable and anticipates congestion using RED, it will set a flag notifying the sender to decrease its congestion window in order to avoid the retransmission of packets.

6. IMPLEMENTATION OF ECN CAPABILITY - IP

6.1 The IP ToS Field

RFC 791 defined the ToS (Type of Service) octet in the IP header; originally this was a field that is used for QoS purposes which consisted of 8 bits, broken into two subfields.

The first subfield (precedence) was used to identify and route packets; the second sub field (ToS) defined the type of service requested for the traffic.

The first two fields of the ToS octet were defined as the "Precedence" and "Type of Service" (ToS) fields. In this RFC, bits 6 & 7 of the ToS octet were "reserved for future use" and as such were set to zero:

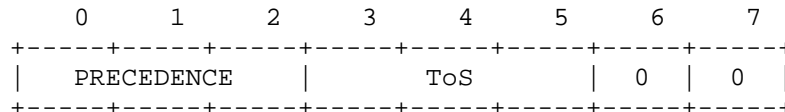


Figure 1: The IP ToS field in RFC 791.

RFC 1122 included bits 6 and 7 in the ToS field but it did not discuss any specific use for those two bits:

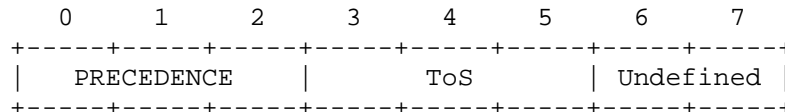


Figure 2: The IP ToS field in RFC 1122.

The IPv4 ToS octet was redefined in RFC with an additional bit added for future ToS types, with the last bit left undefined and set to zero:

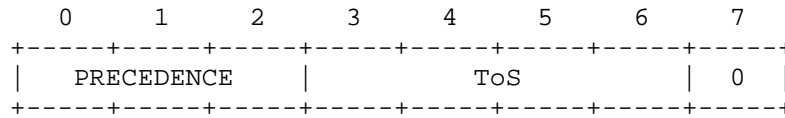
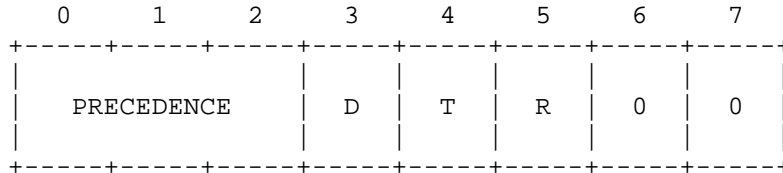


Figure 3: The IP ToS field in RFC 1349.



Precedence (bits 0-2):	Delay/D (bit 3):
111 - Network Control	0 - normal delay
110 - Internetwork Control	1 - low delay
101 - CRITIC/ECP	
100 - Flash Override	Throughput/T (bit 4):
011 - Flash	0 - normal throughput
010 - Immediate	1 - high throughput
001 - Priority	
000 - Routine	
Reliability/R (bit 5):	Reserved/0 (bits 6-7):
0 - normal reliability	Reserved for future use;
1 - high reliability	both set to "0".

Figure 4: The basic IP ToS signalling implementation.

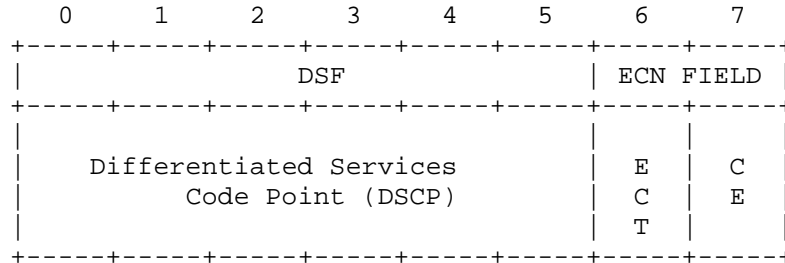
6.2 ECN Implementation - IP

ECN was implemented in IP by the redesignation of the IP ToS (Type of Service) field to the "Differentiated Services Field" and the assignment of two bits of that field for ECN purposes as detailed in RFC 3168.

In the ECN field one bit (ECT - ECN Capable Transport) is used to indicate that an endpoint is ECN capable; the other bit (CE - Congestion Experienced) is used to indicate network congestion.

6.2.1 The Differentiated Services Field

Bits 6 and 7 in the IPv4 octet formerly designated for ToS information is now designated as the IP ECN field in the IP header; the definitions for the IPv4 TOS octet in RFC 791 have been superseded by the six-bit DS (Differentiated Services) Field in RFC 2474 and RFC 2780.



DSF: Differentiated Services Field
 ECN: Explicit Congestion Notification
 ECT: ECN-Capable Transport bit
 CE: Congestion Experienced bit

Figure 5: The implementation of IP ECN within IPv4's former ToS (Type of Service) octet, now designated the "Differentiated Services Field".

6.2.2 IP ECN Field

As of this writing, bits 6 and 7 of the octet are listed in RFC 2474 as "currently unused" and are specified in RFC 2780 as approved for experimental use for ECN; the current ECN field names correspond roughly to the old ones detailed in RFC 2481 as demonstrated in figure 6.

+-----+-----+			
ECN FIELD			
+-----+-----+			
ECT	CE	RFC 2481	RFC 3168
0	0	Not-ECT	Not-ECT
0	1	unused	ECN capable - ECT(1)
1	0	ECN capable	ECN capable - ECT(0)
1	1	CE	CE

Not-ECT packet is not using ECN or endpoint is not ECN capable
 ECT(1) endpoints are ECN-capable
 ECT(0) endpoints are ECN-capable
 CE congestion experienced; indicates end point congestion

Figure 6: The ECN Field in IP.

The ECT bit in the IP ECN field is set by the transmitting routing device in order to indicate that the endpoints of the transport layer protocol (e.g. TCP) are ECN capable; a routing device sets the CE bit when congestion is experienced to indicate congestion to ECN capable endpoints.

A routing device sets the ECT bit only when the packet would have been dropped in order to indicate congestion (e.g. Tail Drop); should the level of congestion exceeds a maximum threshold, then the routing device would drop the packet vice setting the CE bit in order to protect the network from further congestion.

Congested Router Sets the IP CE Bit

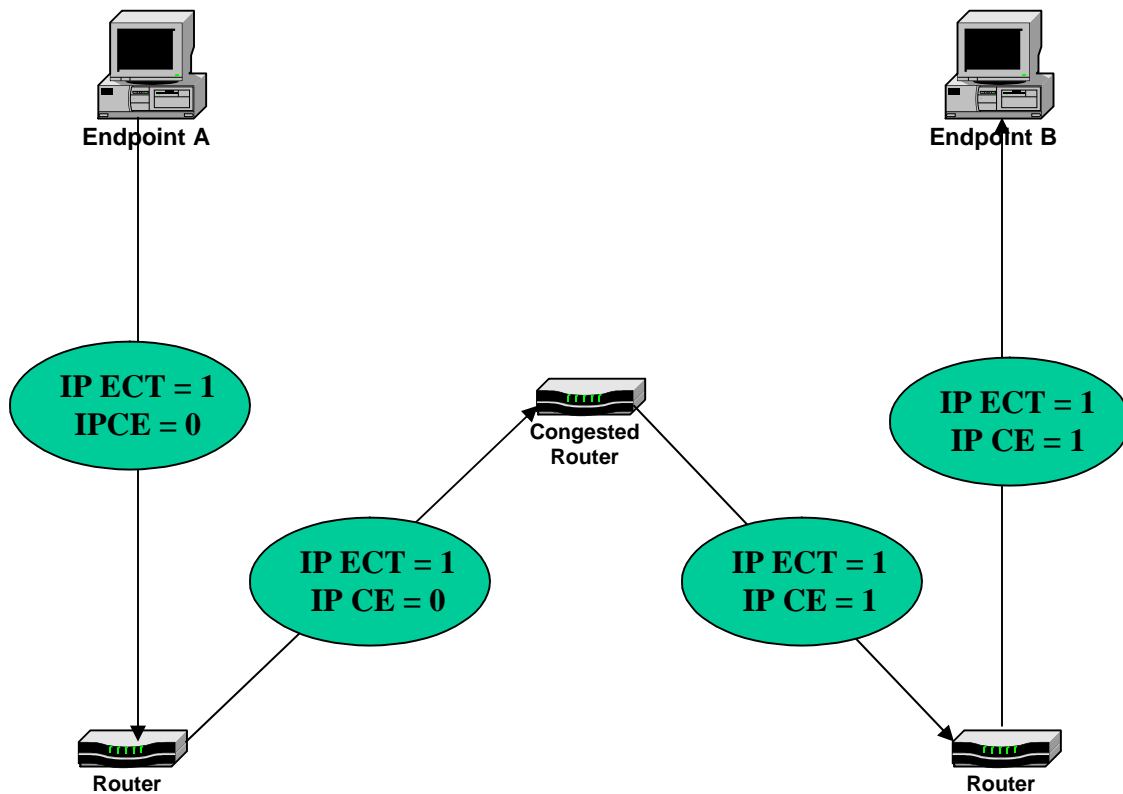


Figure 7: Diagram demonstrating how the IP ECN bits are set by a routing device when congestion is experienced.

7. IMPLEMENTATION OF ECN CAPABILITY - TCP

7.1 CWR/ECE Flag Implementation

RFC 3168 also addressed the implementation of ECN fields in the TCP header in order to provide for ECN negotiation during the TCP connection setup (the three way handshake) as well as providing an ECN echo and reduced congestion window signalling technique.

In order to accomplish this, two additional flag fields (CWR and ECE) were added to bits zero and one in the fourteenth byte of the TCP header, which were originally assigned to the TCP header's "reserved" field.

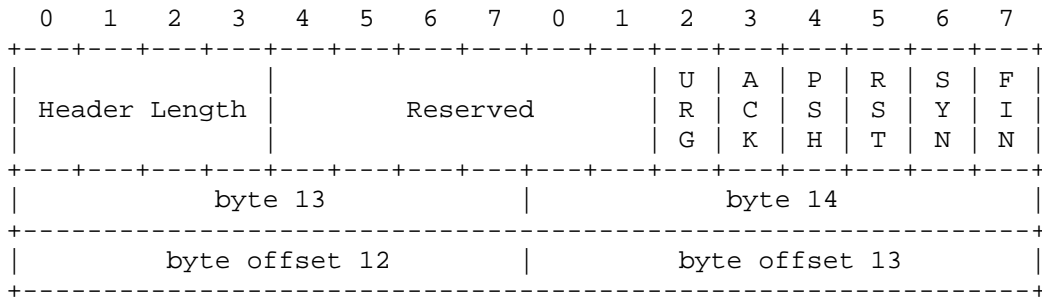


Figure 8: The old definition of bytes 13 and 14 of the TCP header.

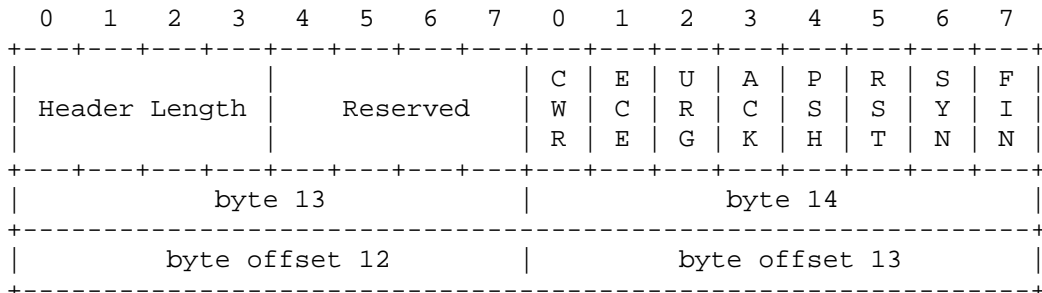


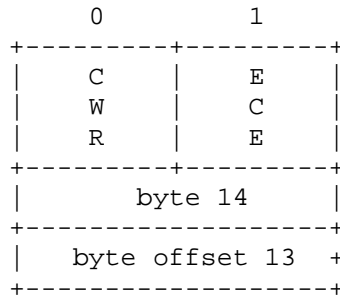
Figure 9: The new definition of bytes 13 and 14 of the TCP header.

7.2.2 TCP ECN Field

As mentioned previously, bits zero and one of the fourteenth byte in the TCP header were assigned to two new flags, CWR (Congestion Window Reduced) and ECE (ECN Echo).

In addition to facilitating ECN negotiation during TCP's three way handshake, the bits can be set to signal certain conditions during traffic operations:

- a. ECE - this bit is set during traffic operations by the receiver of a CE packet (IP header's CE bit set) in order to signal the source of the packet that it has been received; and
- b. CWR - this bit is set during traffic operations so that the source of a CE packet can signal the receiver that the congestion window has been reduced.



CWR: Congestion Window Reduced
ECE: Explicit Congestion Echo

Figure 10: The ECN field in the TCP header.

ECN FIELD
CWR ECE
0 0 Congestion window not reduced/CE packet not received
0 1 Congestion window not reduced/CE packet received
1 0 Congestion window reduced/CE packet not received
1 1 Congestion window reduced/CE packet received

Figure 11: Possible bit settings of the TCP ECN field during traffic operations.

8. EX DUOS UNUM¹⁰ - TCP/IP ECN

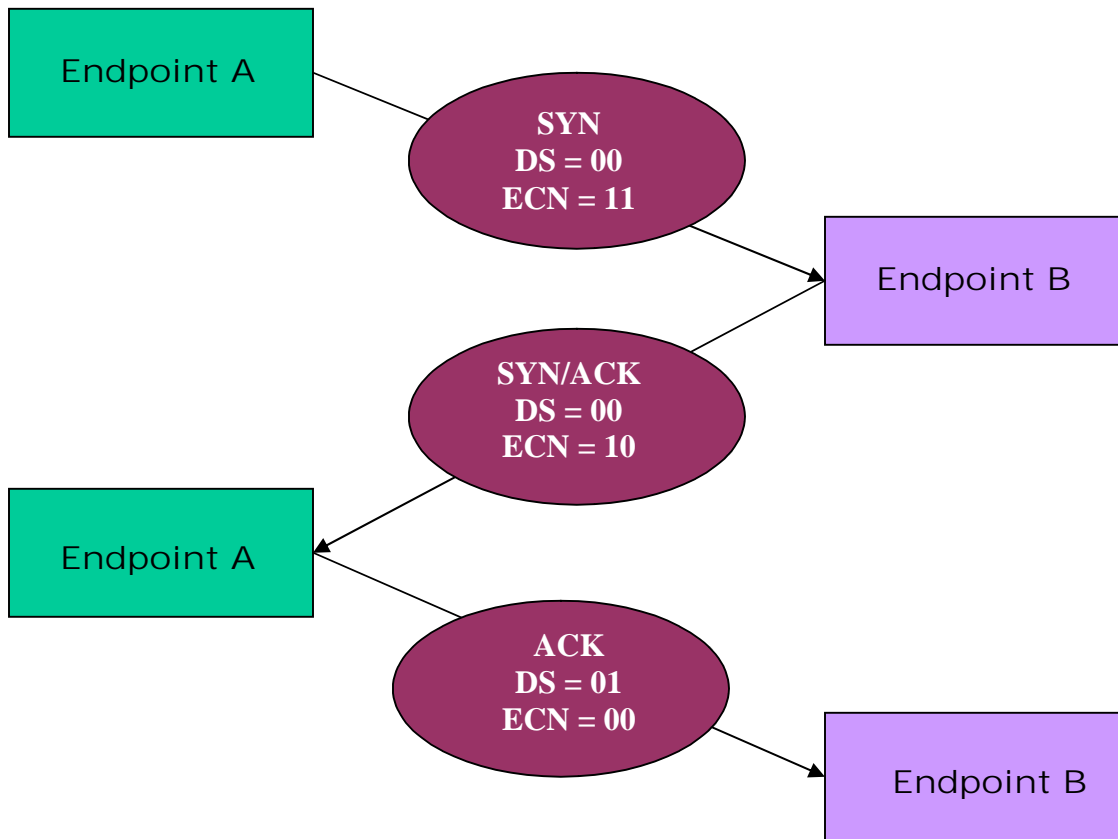
8.1 TCP & IP ECN - WWW Flag Team Champions¹¹

The ECN implementations inherent within TCP and IP function together in order to prevent network congestion over TCP/IP connections.

8.1.1 TCP Three Way Handshake for ECN Capable Endpoints

During the establishment of a TCP connection, ECN capable endpoints can, in addition to exchanging normal information regarding sequence numbers and window sizes, perform ECN negotiation to signal that both are ECN capable as demonstrated below:

ECN Capable Endpoints - TCP Handshake



DS = IP Differentiated Services Field
ECN = TCP Flag ECN Field

Figure 12: Diagram demonstrating ECN negotiation between two ECN capable endpoints during TCP's three way handshake.

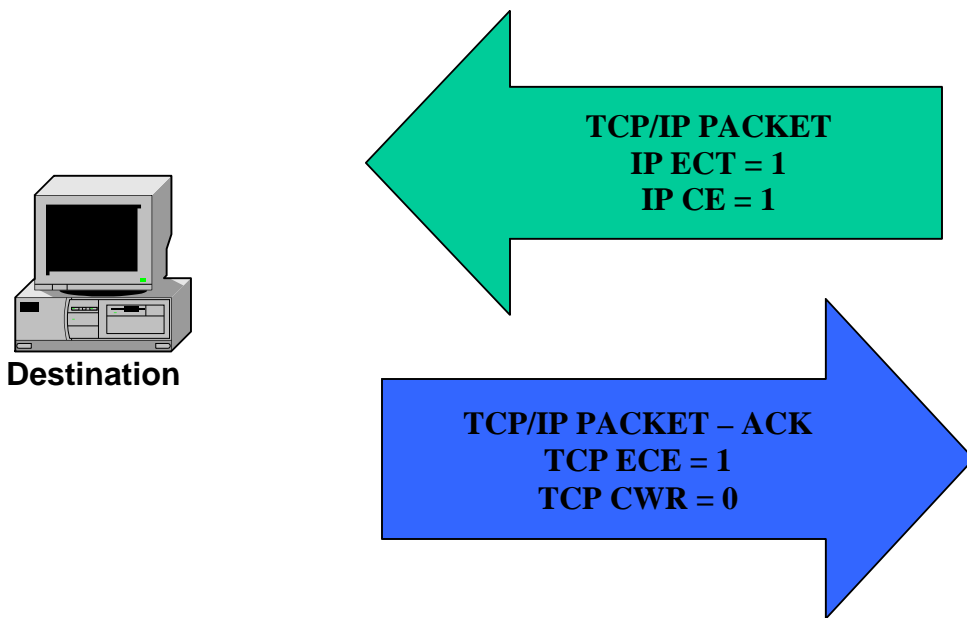
¹⁰ Latin - "From Two, One".

¹¹ Puns intended.

8.1.2 TCP/IP ECN Network Congestion Signalling

Once the TCP/IP stream has been established between two ECN-capable endpoints, the TCP and IP ECN bits can be utilized to signal congestion, perform ECN acknowledgement and reduce the congestion window size.

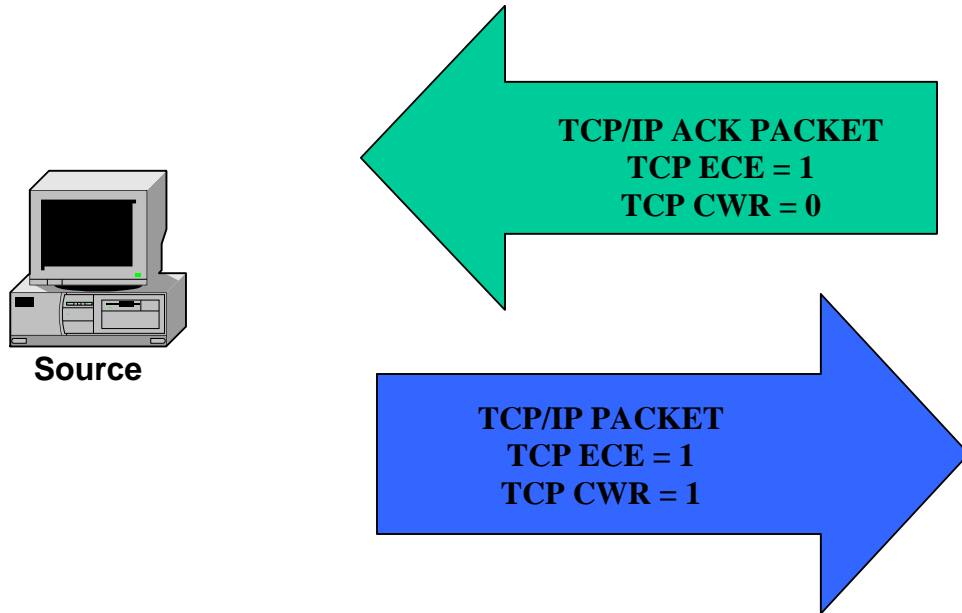
When an ECN capable destination receives a TCP/IP packet with the ECT and CE bits set, the destination transmits a ACK packet with the ECN-Echo bit (ECE) bit set to the source. In order to guard against dropped/last ACK packets, the destination will set the ECN-Echo flag in all ACK packets until such time as a CWR packet is received from the source.



Destination Receives TCP/IP
Packet with IP CE Bit Set to "1"

Figure 13: Diagram demonstrating ECN signalling between ECN capable endpoints when an ECN capable destination receives a TCP/IP packet with the IP CE bit set.

When the source receives the TCP/IP packet from the destination with the ECN-Echo flag set, the source reduces its congestion window size and sets the "congestion window reduced" (CWR) flag of the first packet transmitted to the destination after the window size is reduced.



Source Receives a TCP ACK with TCP ECE Bit Set to "1"

Figure 14: Diagram demonstrating ECN signalling between ECN capable endpoints when an ECN capable destination receives a TCP/IP packet with the IP CE bit set.

Network Congestion Status	IP Differentiated Services Field ECN Bits		TCP ECN Field Bits	
	0	1	0	1
No Congestion	0	1	0	0
No Congestion	1	0	0	0
Congestion	1	1	0	0
Receiving Endpoint Response	1	1	0	1
Transmitting Endpoint Response	1	1	1	1

Figure 15: Diagram demonstrating ECN the possible combinations of ECN bit settings utilized for TCP/IP ECN signalling.

With regards to the scenario demonstrated in figures 13 and 14, the source is not required to retransmit the CE packet for which it received the ECN-Echo ACK because the CE packet arrived successfully at the destination.

In addition, The transmission of the CWR packet is guaranteed because if the packet is dropped, the source will reduce it's congestion window size again and retransmit the dropped CWR packet.

8.2 PROS AND CONS OF ECN IMPLEMENTATIONS

Fortunately, the pros of implementing ECN as the primary congestion control mechanism for TCP/IP communication streams greatly outweigh the cons; the benefits and limitations of are discussed separately in detail below.

8.2.1 Benefits of ECN Based Congestion Avoidance

The benefits of ECN based congestion avoidance implementations are as follows:

- a. like RED, ECN allows active queue memory management;
- b. global TCP synchronization is eliminated;
- c. enhances network's ability to manage traffic surges;
- d. ECN utilizes existing fields of TCP and IP to mark packets, thereby preventing bandwidth consumption caused by the forwarding of additional packets;
- e. ECN greatly reduces transfer times for short TCP flows in which the last packet has been dropped;
- f. ECN can be deployed incrementally without having to partition an existing network into compatible/incompatible segments; and
- g. ECN utilizes in-band signalling techniques, thereby alleviating bandwidth consumption on congested networks.

8.2.2 Limitations of ECN Based Congestion Avoidance

The limitations of ECN based congestion avoidance implementations are as follows:

- a. the implementation of ECN necessitates modifications in deployed transport layer protocols in order to render them ECN capable; and
- b. the uncertain history of the IP ToS byte may result in compatibility problems if use of the IP ECN field is implemented.

9. QUICK REVIEW FOR ANALYSTS - BASIC TCP FLAG ANALYSIS

The following will demonstrate how to calculate which TCP flags are set within the TCP header; this is a rather simple process that will also assist the analyst in the development of bit masks for various flag combinations.

In order to determine which TCP flags are set, first determine the hex value of the flags from the packet capture; in the case below, the TCP flag hex value is "c2", which converts to decimal "194". Once this value is determined, begin subtracting the values of the fields that are less than or equal to the TCP flag decimal value - again in this case, the decimal value is "194":

```
2005/08/22-19:38:05.260832 222.96.156.87.42955 > xxx.xxx.xxx.xxx.22: SWE
3146769132:3146769132(0) win 5840 (DF)
2005/08/22-0x0000      4500 003c 182a 4000 3006 3350 de60 9c57      E.<.*@.0.3P.`.W
2005/08/22-0x0010      xxxx xxxx a7cb 0016 bb8f e2ec 0000 0000      .....
2005/08/22-0x0020      a0c2 16d0 6e54 0000 0204 05b4 0402 080a      ....nT.....
2005/08/22-0x0030      81bb fac6 0000 0000 0103 0300      .....
```

Hex "c2" = decimal "194"
 TCP Window size in bytes = 5840

Figure 16: SWE flagged packet capture.

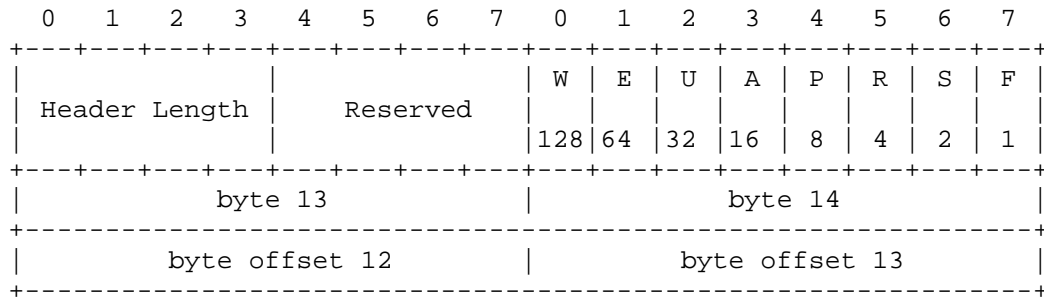
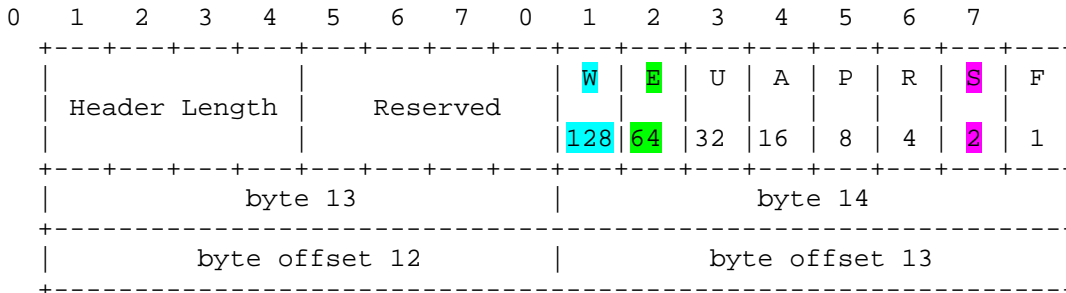


Figure 17: Decimal values assigned to TCP flags



194-128=66 (W/CWR)

66-64=2 (E/ECE)

2-2=0 (S/SYN)

Therefore, the SYN (S), CWR (W) and ECE (E) (thus, "SWE") flags are set in the traffic example demonstrated in figure 16.

Figure 18: Use of decimal subtraction to determine the flags that are set.

Conversely, the process may be used to determine the decimal values to be used for TCP flag bitmasks. For example, to detect packets with the "SWE" flags set, one would use the bitmask demonstrated in figure x.

tcp[13] = 194

tcpheader byte offset argument
 relational operator "equal to"
 decimal value of the TCP flags

Figure 19: TCP bitmask for "SWE" flagged traffic.

The bitmask above, when utilized in a TCPDump filter, will allow the analyst to view all packets with the "SWE" flags set in the TCP header.

For more information regarding the creation of TCPDump filters and bitmasks, please refer to the appendices accompanying this document.

10. APPENDICES

Several appendices have been included in this working aide that provide additional information for the analysts. These include:

- a. "Bit Masking Simplified" - an introduction to writing bitmasks for use in TCPDump filters;
- b. "Byte Offsets for IP, TCP and ICMP Packets" - a working aide that demonstrates popular byte offsets to be used in the development of functional bitmasks; and
- c. "Ted's TCP Dump Filter Collection" - normal people collect coins and stamps but in addition to his computer virus and vintage computer collections, the author collects TCPDump filters - analysts should feel free to utilize them as the need arises.

The appendices can be found in pages A-1 through A-13; analysts are highly encouraged to reference the appendices as required to assist them in the performance of their duties.

A-1. APPENDIX A - "BIT MASKING SIMPLIFIED"

Author: **alt.don**, Posted: Sun Mar 23, 2003 9:03 pm Post subject:
Bitmask usage 101 - TCPdump bitmasking simplified

Bit Masking Simplified

The purpose of bit masking is to allow you to specify specific byte offsets in various protocols ie: parse a large tcpdump file looking for specific flag combinations ie: syn, syn/ack, psh/ack.

Where this would be of benefit is when for example we are looking over a large port 80 scan directed against our networks.. The sheer volume of syn packets plus possible reset packets makes looking over this tcpdump file a chore. One that could result in missing potentially critical packets due to analyst fatigue, and or eye strain. With a proper bit mask in place one can filter many meg's of traffic and whittle it down to a very manageable size showing only psh/ack's for example.

This will help in looking for specific flag combinations for specific ip addresses. In essence it you will save you time allowing you to work more efficiently.

The examples shown below relate only to the tcp header and not icmp, udp, or others. The same theory applies to all the protocols though. One just needs to see what byte offset one wants to filter, and then apply the same concepts described below.

The two IP addresses used here for example purposes will be

10.10.10.100 and 192.168.2.100

The examples shown below as well are done using tcpdump, hence the tcpdump style filters.

Code:

```
-nXvs 0 tcp and host 10.10.10.100 and (tcp[13] & 2 !=0)
```

The above noted breaks out as so;

a) **-nXvs 0** The n means don't convert ip addy's to canonical names ie: leave them in # format. The X means print output in both hex and ascii. The v means to be verbose ie: print out all header info such as ip id numbers and the such. The s means the snaplength. This is the amount of the packet you want to look at. You can put a number after the s or leave it at 0 which will be your default setting.

b) **tcp and host 10.10.10.100** This is where you are specifying that you want to see the tcp protocol and you are specifying as well the host address on which you are running this filter against.

c) `and (tcp[13] & 2 !=0)` This here is the meat of your bit mask. You are using `and` because you are specifying another argument. The `(` tells `tcpdump` this is the beginning of an argument. The `tcp` denotes the `tcp` protocol, and `[13]` denotes the byte offset in the `tcpheader`. The `&` is a primitive allowing you to combine arguments. The `2 !=0` denotes the decimal value in the 13th byte that you want to see. More to follow on what decimal value equates to what flag in the next subpara. The `!=0` means that the bit representing decimal value 2 should be set to 1 ie: the flag is set vice not being set which would be a binary value of 0. The `)` denotes that this is the end of the argument.

d) As mentioned in subpara c above the 13th byte is composed of 8 bits. Each of these bits represents various flags. From the right of the byte the first two bits are assigned to error congestion ie: `ecn` and is not applicable to our bit masking purposes. These first two bits from the right have values of 128 and 64.

The following bit values from the right are:

Code:

URG - ACK - PSH - RST - SYN - FYN

32 16 8 4 2 1

With the values set out above for the 13th byte of the `tcp` header you can now filter out which ever values you wish. Whether they be combinations of flags as will be shown below of simply one flag as shown above.

The below noted example now shows you how to specify several flag in your bit mask.

Code:

```
-nXvs 0 tcp and host 192.168.2.100 and ((tcp[13] & 16 !=0) and (tcp[13] & 8 !=0))
```

The above noted bit mask filter shows you that the `ps`h and `ack` flags for 192.168.2.100 `tcp` traffic will be pulled for. We have two `(` brackets now because we have added a second argument ie: `(tcp[13] & 8 !=0)` So due to this we now need a second bracket to close the argument. You will however get all `ps`h/`ack`'s associated with 192.168.2.100. This may be undesirable if this addy has swept an entire subnet for example. If you want to further refine your search you can specify two specific hosts with the above mentioned bit mask as evidenced below.

Code:

```
-nXvs 0 tcp and host 10.10.10.100 and host 192.168.2.100 and  
((tcp[13] & 16 !=0) and (tcp[13] & 8 !=0))
```

Or you can also do the above with specific ports in mind whether they be source of destination. Please see below.

Code:

```
-nXvs 0 tcp and host 10.10.10.100 and host 192.168.2.100 and dst  
port 80 and ((tcp[13] & 16 !=0) and (tcp[13] & 8 !=0))
```

I just realized that I failed to put in an example showing how you apply bitmasking against a binary file (ie: little endian) Please see the below noted for an example of how you would do it.

Code:

```
tcpdump -r file_name -nXvs 1514 tcp[13] = 18
```

Please note that the value of 18 equates to the decimal value of the flags you want to find. In this case 18 equals both Syn and Ack flags being set in the 13th byte in the TCP header.

The above several noted examples use standard tcpdump filters incorporated with bit masking. The amalgamation of the two will allow you to build complex filters thereby simplifying your task. There are ways of shortening the filters through the use of primitives, however these would be advanced filters. You would need to first be comfortable in the writing of the above before moving on to further complex filters. Should you have any further questions on the above or bit masking in general please feel free to see me or drop me a PM.

Last edited by alt.don on Sun May 02, 2004 2:26 am; edited 2 times in total

A-2. APPENDIX B - "BYTE-OFFSETS FOR IP, TCP AND ICMP PACKETS"

=====
=====

This document is meant to serve as a quick reference for points of interest in IP, TCP, UDP and ICMP headers. I cobbled the information from a variety of sources, all listed at the bottom of this page.

This information will (hopefully) be useful to people building filters for network tools that use BPF, such as tcpdump or snort. I was moved to collect all of this stuff in one place after completing "Intrusion Detection In-Depth" at a recent SANS conference.

Yes, I'm aware that some of these offsets are covered by tcpdump macros. So what? Use the byte offsets instead and let them ph33r your m@d skillz. Corrections, additions and so on are welcome. Send them to:

jquinby (at) node.to

Cheers,

JQ

IP byte offsets

=====
=====

- ip[0] & 0x0f - protocol version
- ip[0] & 0xf0 - protocol options
- ip[0] & 0xff00 - internet header length
- ip[1] - TOS
- ip[2:2] - Total length
- ip[4:2] - IP identification
- ip[6] & 0xa - IP flags
- ip[6:2] & 0x1fff - fragment offset area
- ip[8] - TTL
- ip[9] - protocol field
- ip[10:2] - header checksum
- ip[12:4] - src IP address
- ip[16:4] - dst IP address
- ip[20:3] - options
- ip[24] - padding

Src IP = Dest IP (land attack)
(ip[12:4] = ip[16:4])

IP versions !=4
(ip[0] & 0xf0 != 0x40)

IP with options set:
(ip[0:1] & 0x0f > 5)

Broadcasts to x.x.x.255:
(ip[19] = 0xff)

Broadcasts to x.x.x.0
(ip[19] = 0x00)

TCP byte offsets, including anomalous TCP flag settings.

=====
=====

```
tcp[0:2]          - src port
tcp[2:2]          - dst port
tcp[4:4]          - seq number
tcp[8:4]          - ack number
tcp[12] & 0x00ff - data offset
tcp[12] & 0xff00 - reserved
tcp[13]          - tcp flags
tcp[13] & 0x3f = 0 - no flags set (null packet)
tcp[13] & 0x11 = 1 - FIN set and ACK not set
tcp[13] & 0x03 = 3 - SYN set and FIN set
tcp[13] & 0x05 = 5 - RST set and FIN set
tcp[13] & 0x06 = 6 - SYN set and RST set
tcp[13] & 0x18 = 8 - PSH set and ACK not set
tcp[13] & 0x30 = 0x20 - URG set and ACK not set
tcp[13] & 0xc0 != 0 - >= one of the reserved bits of tcp[13] is set
tcp[14:2]        - window
tcp[16:2]        - checksum
tcp[18:2]        - urgent pointer
tcp[20:3]        - options
tcp[23]          - padding
tcp[24]          - data
```

UDP byte offsets, header only

=====
=====

```
udp[0:2]          - src port
udp[2:2]          - dst port
udp[4:2]          - length
udp[6:2]          - checksum
udp[8:4]          - first 4 octets of data
```

Crafted packets with impossible UDP lengths:

udp[4:2] < 0) or (udp[4:2] > 1500

ICMP

=====
=====

```
icmp[0]           - type
icmp[1]           - code
icmp[3:2]         - checksum
```

Destination Unreachable:

```
icmp[0] = 0x3 (3)
icmp[4:4]         - unused (per RFC)
icmp[8:4]         - internet header + 64 bits original data
icmp[1]           - 0 = net unreachable;
                  - 1 = host unreachable;
                  - 2 = protocol unreachable;
                  - 3 = port unreachable;
                  - 4 = fragmentation needed and DF set;
                  - 5 = source route failed.
```

Time Exceeded:

icmp[0] = 0xB (11)

icmp[4:4] - unused (per RFC)
icmp[8:4] - internet header + 64 bits original data
icmp[1] - 0 = TTL exceeded intransit
- 1 = fragment reassembly time exceeded

Parameter Problem:

icmp[0] = 0xC (12)

icmp[1] - 0 = pointer indicates error
icmp[4] - pointer
icmp[5:3] - unused, per RFC
icmp[8:4] - internet header + 64 bits original data

Source Quench:

icmp[0] = 0x4 (4)

icmp[1] - 0 = may be received by gateway or host
icmp[4:4] - unused, per RFC
icmp[8:4] - internet header + 64 bits original data

Redirect Message:

icmp[0] = 0x5 (5)

icmp[1] - 0 = redirect for network
- 1 = redirect for host
- 2 = redirect for TOS & network
- 3 = redirect for TOS & host
icmp[4:4] - gateway internet address
icmp[8:4] - internet header + 64 bits original data

Echo/Echo Reply:

icmp[0] = 0x0 (0) (echo reply)

icmp[0] = 0x8 (8) (echo request)

icmp[4:2] - identifier
icmp[6:2] - sequence number
icmp[8] - data begins

Timestamp/Timestamp Reply:

icmp[0] = 0xD (13) (timestamp request)

icmp[0] = 0xE (14) (timestamp reply)

icmp[1] - 0
icmp[4:2] - identifier
icmp[6:2] - sequence number
icmp[8:4] - originate timestamp
icmp[12:4] - receive timestamp
icmp[16:4] - transmit timestamp

Information Request/Reply:

icmp[0] = 0xF (15) (info request)
icmp[0] = 0x10 (16) (info reply)

icmp[1] - 0
icmp[4:2] - identifier
icmp[6:2] - sequence number

Address Mask Request/Reply:

icmp[0] = 0x11 (11) (address mask request)
icmp[0] = 0x12 (12) (address mask reply)

Sources:

RFC768, "User Datagram Protocol Specification"
RFC791, "Internet Protocol Specification"
RFC792, "Internet Control Message Protocol Specification"
RFC793, "Transmission Control Protocol"
filter files from SHADOW-1.8 source distribution
man pages for tcpdump
"TCP/IP and tcpdump Pocket Reference Guide", SANS

ADDENDUM: *This appendix has been reproduced on the our shared network folder in order to facilitate analysts in the performance of their duties; the document can be found in the path designated \working_aides\.*

A-3. APPENDIX C - "TED'S TCPDUMP FILTER COLLECTION"

For your convenience, I hereby present some of the filters I've either acquired, written or borrowed over the last eight years or so.

Corrections, additions, etc. are of course welcome; the analysts should feel free to add their own filters if they haven't been covered here. Happy packet sniffing!

NOTE: *Some of these filters have never been tested in our live network environment and may require tinkering.*

=====
=====

Active open packets (syn is set, ack is not)
tcp[13] & 2 != 0) and (tcp[13] & 0x10 = 0)

Bad event filter - may detect potential threats
tcp and (tcp[13] & 0x02 != 0) and (tcp[13] & 0x10 = 0)and (not dst port 53) and (not dst port 80) and (not dst port 25) and (not dst port 21)

Back Orifice Trojan 1
udp && dst port 31337

Back Orifice Trojan 2
udp and dst port 31337

Broadcasts to x.x.x.255
ip[19] = 0xff

Broadcasts x.x.x.0
ip[19] = 0x00

DNS AXFR (zone transfer)
tcp && dst port 53

DNS queries/responses (usually UDP)
udp and dst port 53

Exec
dst port 512

Fragments
ip[6:2] & 0x2000 != 0

Fragmentation attack
tcp && ip[6:2]&16383 != 0

Destination ports less than 20
tcp[2:2] < 20

ELEET (hacker signature port)
(dst port 31337) or (src port 31337)
Epmmap - DCE Endpoint resolution
dst port 135

Finger
dst port 79

FIN flag set/ACK flag not set
tcp[13] & 0x11 = 1

Fragmented packets with more coming
ip[6:1] & 0x20 != 0

Fragments with more coming
ip[6:1] & 0x20 !=0

Fragments with zero offset
ip[6:2] & 0x1fff = 0

Fragments bit is not set, fragment offset is not zero
((ip[6:1] & 0x20 = 0) && (ip[6:2] & 0x1fff != 0))

Gnutella p2p - filter out Gnutella packets
tcp[(tcp[12]>>2):4] = 0x474e5554 && \
tcp[(4+(tcp[12]>>2)):4] = 0x454c4c41 && tcp[8+(tcp[12]>>2)] = 0x20

Gopher
dst port 70

Header more than 20 bytes.
ip[0] & 0x0f > 5

Hostile SNMP
udp port 161 or udp port 162) and not src net x.x

ICMP
icmp and icmp[0] != 8 and icmp[0] != 0

ICMP - Error messages: fragmentation needed but DF flag set 1
(icmp[0] = 3) and (icmp[1] = 4)

ICMP - Error messages: fragmentation needed but DF flag set 2
((icmp[0] = 3) && (icmp[1] = 4))

ICMP - Error messages: source route failed (look for outbound)
(icmp[0] = 3) and (icmp[1] = 5)

ICMP - Fragmented packets (look for inbound)
icmp and (ip[6:1] & 0x20 != 0)

ICMP - ICMP packets that are not ping packets
icmp and icmp[0] !=8 and icmp[0] != 0

IP options set 1
(ip[0:1] & 0x0f) > 5

IP options set 2
(ip[0] & 0x0f) != 5

IP versions than ipv4
ip && (ip[0] & 0xf0 != 0x40)

IRC

tcp and port 6667

IRC (busy servers)

(dst port 6667) and (dst port 6668) and (dst port 6669) and (dst port 6670)

IRC - TCP port 6667 with ACK flag set and payload starting at byte 12 that does not include the ascii "PING", "PONG", "JOIN", or "QUIT".

```
(tcp[13] & 0x10 = 1) && (tcp[0:2]=6667 || tcp[2:2]=6667) \  
&& (not ip[32:4] = 1346981447 || not ip[32:4] = 1347374663 \  
|| not ip[32:4] = 1246710094 || not ip[32:4] = 1364543828)
```

IMAP

tcp and (tcp[13] & 2 != 0) and (dst port 143)

IMAP attacks (SYN)

tcp and (tcp[13] & 2 != 0) and (dst port 143)

Kerberos admin

dst port 749t

Land attacks 1

ip[12:4] = ip[16:4]

Land attacks 2

(tcp[0:2] = tcp[2:2]) && (ip[12:4] = ip[16:4])

Link (hacker signature port)

(dst port 87) or (src port 87)

Listen - the System V listener

dst port 2766

Loadav

dst port 750

Login

dst port 513

Loki echo request or reply

```
icmp and (icmp[0] = 8 or icmp[0] = 0) and (icmp[6:2] = 0xf001 or  
icmp[6:2] = 0x0af0)
```

Loose source routing

ip[20:1] & 0xff = 131

NetBus

tcp and (dst port 12345) or (dst port 12346)

NetBIOS datagram service

dst port 138

NetBIOS name service

dst port 137

NetBIOS session service
dst port 139

NeWS window system
dst port 144

NFS
ip and udp port 2049

Nmap OS fingerprinting - ACK flag set, ack value is zero.
tcp[13] & 0xff = 0x10 && tcp[8:4] = 0

Nmap OS fingerprinting - high-order reserved bits not zero.
tcp[13] >= 64

NNTP
tcp and port 119

Null scans 1
tcp[13] = 0

Null scans 2
tcp[13] & 0xff = 0

Openwin (like X11)
dst port 2000

Payload of more than 20 bytes
(ip[2:2] - ((ip[0]&0x0f)<<2) - (tcp[12]>>2)) <= 20

ping of death attack
((ip[6] & 0x20 = 0) && (ip[6:2] & 0x1fff != 0)) && \
((65535 < (ip[2:2] + 8 * (ip[6:2] & 0x1fff))

Portmapper
ip and dst port 111

Printer spooler (look for unusual activity)
dst port 515

PROFILE naming system
dst port 136

Pump
dst port 751

PSH and ACK flags set
tcp and (tcp[13] & 8 !=0) and (tcp[13] & 0x10 !=0)

RST and FIN flags set
tcp[13] & 0x05 = 5

Shell
dst port 514

SMB
dst port 139 && tcp[13:1] & 18 = 2

SNMP

(udp port 161 or udp port 162) and not src net 172.17

Socks

tcp and (tcp[13] & 2 !=0) and dst port 1080)

Source route failed

icmp and (icmp[0] = 3) and (icmp[1] =5)

Source ports less than 20

tcp[0:2] < 20

Suupdp (hacker signature port)

(dst port 95) or (src port 95)

SYN and ACK flags set

tcp and (tcp[13] & 2 !=0) and (tcp[13] & 0x10 !=0)

SYN and ACK flag not set

tcp and (tcp[13] & 2 !=0) and (tcp[13] & 0x10 =0)

SYN and FIN flags are set simultaneously

(tcp[13] & 2 != 0) and (tcp[13] & 1 !=0)

SYN/FIN scans

tcp[13] = 3

SYN flag set, high-traffic ports filtered

tcp and (tcp[13] & 0x02 != 0) and (tcp[13] & 0x10 = 0)and (not dst port 53) and (not dst port 80) and (not dst port 25) and (not dst port 21)

SYN flag only set

tcp[13] & 0x02) != 0

TCP packets except ACK/PSH

(tcp[13] & 0xe7) != 0

TCP packets with reserved bits set

tcp[14] >= 64

Teardrop attacks 1

ip[6:1] & 0x20 != 0

Teardrop attacks 2

udp && (ip[6:1] & 0x20 != 0)

Telnet

dst port 23

TTL less than 5 (possible Firewalking)

ip[8] < 5

Traceroute destination ports 1

(udp[2:2] >= 33000) && (udp[2:2] <= 33999)

Traceroute destination ports 2

udp[2:2] >= 33000 && udp[2:2] < 34000 && ip[8] = 1

Traceroute destination ports 3
udp and (udp[2:2] >= 33000) and (udp[2:2] <33999)

UDP destination ports less than 20
udp[2:2] < 20

UDP packets with impossible udp lengths
(udp[4:2] < 0) || (udp[4:2] > 1500)

UDP source ports less than 20
udp[0:2] < 20

Unknowns (please notice the "not")
not ((tcp[2:2] < 20) or dst port 21 or dst port 23 or dst port 22 or
dst port 25 or dst port 37 or dst port 43 or dst port 53 or dst port 70
or dst port 79 or dst port 87 or dst port 95 or dst port 109 or dst
port 110 or dst port 111 or dst port 113 or dst port 119 or dst port
135 or dst port 136 or dst port 137 or dst port 138 or dst port 139 or
dst port 143 or dst port 144 or dst port 443 or dst port 512 or dst
port 513 or dst port 514 or dst port 515 or dst port 540 or dst port
563 or dst port 666 or dst port 749 or dst port 750 or dst port 751 or
dst port 1080 or dst port 1352 or dst port 1452 or dst port 1494 or dst
port 1521 or dst port 1526 or dst port 2000 or dst port 2766 or dst
port 6667 or dst port 31337 or dst port 80 or dst port 8000 or dst port
8080)

Unroutable IP addresses (0.x.x.x, 127.x.x.x, 1.x.x.x, 2.x.x.x, 5.x.x.x
or 240.x.x.x thru 255.x.x.x)
Respectively: net 0, net 127, net 1, net 2, net 5 or ip[12] > 239

URG flag is set
tcp[13] & 0x20 !=0

UUCP
dst port 540

Whois
dst port 43

X11 filter for X11 or Motif traffic:
tcp and (port 6000 or port 6001 or port 6002)

X11 ports
(tcp[2:2] >= 6000) and (tcp[2:2] < 7000)

*NIX R-utilities
ip and (tcp dst port 512 or tcp dst port 513 or tcp dst port 514)

ADDENDUM: *This appendix has been reproduced on the our shared network folder in order to facilitate analysts in the performance of their duties; the document can be found in the path designated \working_aides\.*

Sources & References

- Allman, M. & Paxson, V. & Stevens, W. (April 1999). RFC 2581. *TCP Congestion Control*. Retrieved August 20, 2005, from <ftp://ftp.rfc-editor.org/in-notes/rfc2581.txt>
- Almquist, P. (July 1992). RFC 1349. *Type of Service in the Internet Protocol Suite*. Retrieved September 07, 2005, from <http://www.faqs.org/rfcs/rfc1349.html>.
- Alvin, C. & Fisher S. & Hardy, M. & Rodin, J. & Smith, N. & Wheeler D.A., et al. (July 2005). Wikipedia. *Network Congestion Avoidance*. Retrieved August 21, 2005, from http://en.wikipedia.org/wiki/Explicit_Congestion_Notification
- Andrews, M. & Black, D. & Floyd, S. & Ramakrishnan, K. (September 2001). RFC 3168. *The Addition of Explicit Congestion Notification (ECN) to IP*. Retrieved August 20, 2005, from <ftp://ftp.rfc-editor.org/in-notes/rfc3168.txt>
- Bubenius, C. & Burnett, C. & Cardwell, N., & Green, P. & Sidwell, R., et al. (August 2005). Wikipedia. *Transmission Control Protocol*. Retrieved August 20, 2005, from http://en.wikipedia.org/wiki/Transmission_Control_Protocol
- Cisco Systems Inc. Cisco Documentation. *Congestion Avoidance Overview*. Retrieved August 23, 2005, from http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/qos_c/qcpart3/qcconavd.htm#23395
- Floyd, S. (May 2004). *ECN (Explicit Congestion Notification) in TCP/IP*. Retrieved August 20, 2005, from <http://www.icir.org/floyd/ecn.html>
- Jacobson, V. & Karels, M. (November 1988). *Congestion Avoidance and Control*. Retrieved August 20, 2005, from <http://www-nrg.ee.lbl.gov/papers/congavoid.pdf>
- Parker, D.D. (alias "alt.don"). (2003, March 23). Security Forums. *Bit Masking Simplified*. Posted to <http://www.security-forums.com/forum/viewtopic.php?t=4489>
- Quinby, J. (Date unkown). *Byte-offsets for IP, TCP and ICMP Packets*. Retrieved August 19, 2005, from http://packet.node.to/hacks/byte_offsets.txt
- Semeria, C. (February 2002). Juniper Networks. *White Paper - Supporting Differentiated Service Classes: Active Queue Memory Management*. Retrieved August 19, 2005, from http://www.juniper.net/solutions/literature/white_papers/200020.pdf