Bandwidth management and optimisation

Policy development workshop

Unit 1

Hand-out: Making sense of traffic graphs

In this document we show a sample traffic graph and describe the key TCP/IP concepts that participants need to be able to read traffic graphs.

Document Notes

|  |  |
| --- | --- |
| Author: | TENET, Duncan Greaves; INASP, Manuela Bianco |
| File name: | Unit1-handout.docx |
| Date submitted: | 20/08/2013 11:47 |
| Date last updated: | 21/08/2013 12:38 |

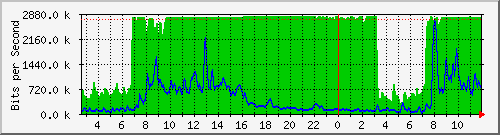
Summary

On completion of this session the learner will have:

1. Understood how to read traffic graphs

Making sense of traffic graphs

The following traffic graph is a typical case of a saturated circuit, graphed over about 36 hours. The green part of the graph shows the inbound traffic; the blue part, the outbound. The outbound traffic is not saturated at all, but as we’ll see in a moment, this doesn’t mean that outbound traffic is free of problems. The inbound traffic is at saturation point from about seven o’clock in the morning until three o’clock the next morning.



[Note; when printed in black and white, the shaded area is the green area and the line is blue]

To understand why a graph like this is such a problem, we need to understand a bit about TCP/IP.

TCP/IP is actually two protocols which work together: TCP and IP, hence the name. IP – the Internet Protocol – is the basic protocol on which the Internet is built. IP is a “connectionless” and “unreliable” protocol. IP is “connectionless” because routers send IP datagrams (the chunks of data with addressing information that the protocol uses) without any knowledge of whether they have been delivered or not, or indeed any interest in the issue: IP datagrams are like arrows fired in the air. And IP is “unreliable” not in the idiomatic sense, but in the sense that datagrams may be lost, delayed, fragmented, or arrive out of order, and the delivery is a “best effort” service rather than a guaranteed one. IP is in fact amazingly reliable under ordinary circumstances: unreliable is a technical rather than a descriptive term.

Applications don’t use IP directly, except in very rare cases. Generally they use another protocol which in turn uses the IP protocol. In most cases – certainly for the two core applications of the Internet, mail and the web – they use a protocol called TCP, which is the Transmission Control Protocol. TCP adds (among other things) “flow control” to the IP layer – it is a connection-oriented, reliable-transport protocol. It is connection-oriented because it establishes a state of connectedness between two hosts (i.e. machines, whether client or server); and it provides reliable transport, because all the data in a TCP session is guaranteed either to be delivered or, if it isn’t, to make sure that error messages go to the application using it. TCP works with IP by creating chunks of data called “segments”, with rich flow control information inside them, and these are then packaged inside IP datagrams and sent.

One of the purposes of flow control is to allow TCP to adjust dynamically to changing bandwidth conditions, and, in particular, to respond to congestion by slowing down the rate of transmission. TCP starts up in “slow start” mode, and, if the bandwidth is available, very quickly ramps up to much faster transmission. As TCP segments are received so they are acknowledged, and the sending host knows it can send more. (This is something of a simplification – more than one segment is sent at a time, depending on the “window size” that the receiving host advertises, but it will do for our purposes.) If a segment is not acknowledged then it is retransmitted; if segments are not received as expected, then the host will re-enter “slow start” mode, or it will break off the connection entirely – a “TCP reset”.

Now consider the traffic graph above. Many more IP datagrams are being directed down this circuit than it has the capacity to carry. (Remember that TCP segments are carried inside IP datagrams.) Of course, it is possible that the demand on the circuit is exactly equal to what it can supply, but this is most unlikely, unless the institution in question is running sophisticated bandwidth management software, and even then a circuit is never allowed to fill to this extent. Datagrams that can’t be transmitted are placed in router queues, but queues are not infinite; eventually they overflow, and then datagrams are discarded. (In fact, long before a queue overflows, the router will probably start discarding datagrams, not necessarily from the end of the queue, according to a probabilistic protocol.)

For a web browser to load a web page from a server, it must open a TCP session for each object on the web page. A typical web page is 18 objects, so 18 TCP sessions are needed. Each TCP session that completes normally means that that object is properly received and can be displayed on the page. If a TCP session never gets out of slow start, or if its performance is badly degraded by transmission delays, then that object will take a long time to load. If a TCP session breaks completely before the object is retrieved then the objects is lost entirely – it might displayed as a missing image, or, if it is critical to the architecture of the page (for example, it is the underlying html code) then the browser will display nothing except an error. If the object is a single large entity, such as a PDF file or a large scanned image, then it is much more vulnerable to TCP degradation than a small object, and it is much more likely to be a victim of a broken session (because it takes more TCP segments to transmit, and hence a longer TCP session.)

Now consider the graph again. The typical user experience here is: slow transmission, and probably lots of TCP resets. Users respond to resets by trying again, causing the same data to flow all over again, and possibly to get lost all over again. In addition, segment retransmissions are also consuming bandwidth unnecessarily – the same segment flows twice, but is only used once. A circuit in this condition is performing well below 100% efficiency, even though it is 100% utilized.

Some users respond to congestion by resorting to tools that are designed to work even under extreme congestion. These tools work by generating even more waste, making the problem worse for everyone, but eventually delivering the object for the user, though at massive cost in wasted bandwidth. Other users will respond to poor performance by opening multiple windows and setting numerous page loads running – once again, some stuff will get through but the underlying problem becomes even more aggravated. In general, the users who resort to these tactics have significant amounts of discretionary time on their hands and are probably in pursuit of content for private gratification. Delay pools cut these users down to size by giving them a fixed amount of bandwidth, and all the applications they run must share it: multiple windows and jackhammer-type tactics are ineffective under these circumstances.

A well-managed connection runs below its theoretical maximum capacity. Bandwidth management tools allow an institution to take utilization levels relatively close to maximum but without causing datagram loss.