Testing of DCCP at the Application Level

Richard Hughes-Jones[*]
The University of Manchester
E-mail: `R.Hughes-Jones@manchester.ac.uk`

Stephen Kershaw
The University of Manchester
E-mail: `Stephen.Kershaw@manchester.ac.uk`

Datagram Congestion Control Protocol (DCCP) is a recently developed transport protocol whose development and implementation in Linux is being aided by the work of Mark Handley and Andrea Bittau in the ESLEA project. The protocol is attractive to many applications where data is transferred with tight constraints on the timing of data delivery, such as internet telephony and e-Science applications such as e-VLBI.
Porting test programs to DCCP has allowed the investigation of the DCCP implementation in recent releases of the Linux kernel and reporting of performance test results. A suggested approach for the use of DCCP for e-VLBI is discussed, with a proposal for a new CCID.

Lighting the Blue Touchpaper for UK e-Science - Closing Conference of ESLEA Project
March 26-28, 2007
Edinburgh

[*]Speaker.

1. Introduction

Datagram Congestion Control Protocol (DCCP) is a recently developed transport protocol, similar in parts to both TCP and UDP with the intention that certain applications and the transport of certain types of data may benefit. Where congestion control is required but reliability is not, DCCP provides a transport level option attractive to many applications such as VoIP and e-VLBI. The congestion control algorithm, CCID, used by DCCP is selectable, allowing DCCP to be tuned more closely to the requirements of a particular application. CCID2 is *TCP-like Congestion Control*, closely emulating Reno TCP while CCID3 is *TCP-friendly rate control*, minimising rate fluctuations whilst maintaining long-term TCP friendly behaviour.

DCCP has been in the Linux kernel since 2.6.14, with recent kernel releases such as 2.6.19 and 2.6.20 having an implementation, incorporating the code developed by ESLEA, that is often considered as fairly stable and high-performance. We report on the porting of a network testing application to DCCP, experiences with creating a stable DCCP testbed and results from initial performance tests.

2. Porting of test software

In order to test the performance of DCCP, software tools were required hence *DCCPmon* is a port of *UDPmon* by the original author, Richard Hughes-Jones [1]. Guidance was given by Andrea Bittau to help with the port to DCCP and the resulting application is being used and proving to work well. However, the process was not entirely trouble-free - some problems were encountered that were perhaps indicative of an implementation that is in development, rather than complete and polished. DCCP related #defines were not to be found in the userland include files, an issue mitigated by creating specific include files. Some system calls were noted to be missing and the API was in a state of flux with functions changing between kernel releases 2.6.19 and 2.6.20. For this reason, and due to limited testing, *DCCPmon* is currently still considered by the author as experimental.

During the development of *DCCPmon* and for corroboration of results, a patched version of *iperf* [2] was used. In addition to the information from the main test application it is desirable to gather data from as many other sources as possible. One useful window into the kernel networking stack is though the kernel SNMP statistics, however there are currently (as of kernel 2.6.21) no SNMP counters for DCCP variables. These statistics would also have been invaluable when problems became apparent with certain kernel versions and it would certainly be a worthy addition to the implementation at the earliest opportunity.

3. End-host setup

The computers used as end-hosts were server-quality SuperMicro machines, with all configurations tested to give 1 Gbit/s throughput using UDP/IP or TCP/IP over Gigabit Ethernet interfaces. The systems used Intel Xeon CPUs and were running Scientific Linux or Fedora distributions of Linux. We had systems using two Dual Core Intel Xeon Woodcrest 5130 CPUs clocked at 2 GHz, dual-booting 32-bit and 64-bit distributions of Fedora Core 5. We also had systems with two Intel

Xeon 2.4 GHz Hyper-Threaded CPUs using a 32-bit distribution of Scientific Linux 4.1. All systems were equipped with and DCCP tested with on-board Intel e1000 Gigabit Ethernet ports. Tests with UDP and TCP gave stable line-rate performance over all tested networks, including 1 Gbit/s over a transatlantic lightpath.

4. Experiences with the Linux DCCP implementations

While developing the *DCCPmon* program and preparing for performance tests, several different Linux kernels have been used, often displaying undesirable effects. With such a new implementation of a new protocol it has often been unclear whether we are seeing problems with the DCCP implementation or something specific to our systems, however we report on our findings and some of the steps taken to achieve a stable DCCP test bed.

4.1 Kernel version 2.6.19-rc1

This kernel version is a release candidate for stable kernel version 2.6.19, which was tested before the stable kernel version was released. Using both *DCCPmon* and *iperf* it was found that we were not getting a working DCCP connection - *tcpdump* showed that the connection was successfully made, with packets exchanged both ways but no ACKs were sent in response to data packets received. In the absence of feedback the sender-side DCCP transmit timer progressively fell back until a threshold upon which DCCP terminated the connection.

We conducted many diagnostic tests to establish the cause of the problem. Advanced features of the network interface card were disabled and DCCP data was sent though a tunneled connection to prevent possible discrimination of the new protocol. Eventually, inserting debugging code into the kernel showed that data were incorrectly being discarded due to header checksum errors, a problem that was later fixed in the network development tree and merged into the stable 2.6.19 kernel release.

4.2 Kernel versions 2.6.19 and 2.6.20

As previously noted, the API calls changed slightly, necessitating further development of the test software code, after which, with the checksum problems resolved it was hoped that interesting tests could be run.

The initial results were promising, with CCID2 showing short-term line-rate throughput - a useful data rate of around 940 Mbit/s after header overheads. CCID3 had an average rate of around 300 Kbit/s but unfortunately DCCP proved to be unstable using either CCID on our 64-bit systems. Transfers would often only last for a few seconds before the receiving system hung with a kernel panic. Some tests would continue for longer, a few minutes with the same throughput performance, but all would trigger a kernel panic within four minutes and repeating tests with larger packet sizes would lead to a quicker crash. The crash dumps associated with the panic generally indicated that the crashes were occurring most regularly in the region of the packet reception code of the network interface card (NIC), where memory is allocated to store incoming packets.

Repeating the tests using a 32-bit distribution and kernel on the same computers yielded the same behaviour, however the older systems running Scientific Linux on Hyper-Threaded Xeon processors proved to be more stable, with extended runs possible, with the majority of transfers persisting until deliberately terminated after many tens of minutes. The system logs, however, showed that everything was not perfect, with many zero order page allocation failures logged, in a similar context to the panics - close to the receive interrupt of the NIC.

5. Towards a stable test bed

Analysis of crash dumps and kernel messages, showed that most error messages were generated when memory was being allocated in NIC RX IRQ handler. To attempt to fix the problem the operation of the NIC driver was analysed together with aspects of the kernel memory management code.

In general, when a request is made for memory allocation, the request will either be serviced immediately (if memory is available) or it will be blocked while sufficient memory is reclaimed. However, when memory allocation is requested in an interrupt context, for example memory allocation to store received packets, blocking is forbidden. In order that the memory allocation has a higher chance of succeeding, the kernel reserves some memory specifically for this situation where the allocation is classed as *atomic*. The amount of memory reserved for atomic allocations is determined by the value of the *min_free_kbytes* sysctl variable.

Increasing the *min_free_kbytes* parameter in the receiving host from the default value of 5741 to 65535 proved to prevent all the previously seen error messages, though it is not entirely clear to us why the memory allocation problems originally occur. It is possible that the default value of *min_free_kbytes* is not sufficient relative to the time between scheduled runs of the memory management daemon (e.g. kswapd), which are scheduled to keep that minimum amount of memory free. A larger value of *min_free_kbytes* may mean that the reserved memory is never filled before the memory management routines can be run. As we do not encounter similar problems with UDP and TCP, it is possible that the higher CPU utilisation of DCCP could cause such a situation by using more CPU time. It is strange that on one system the allocation failures prompted errors messages while on another the result was a fatal system crash.

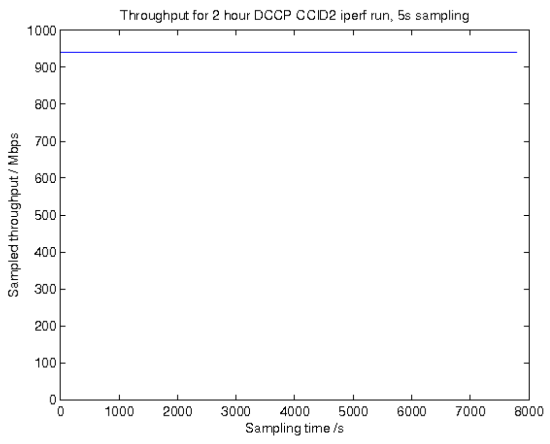
The problem is not entirely mitigated though as even with the increased value of *min_free_kbytes* crashes persist if the packet size is increased sufficiently. More investigation is needed to gain a full understanding of this unwanted feature of our DCCP test.

6. Results of recent tests

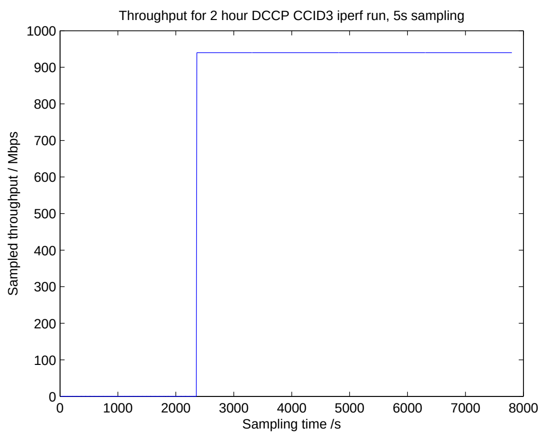
With an increased value of *min_free_kbytes* on the receiving hosts, the systems proved to be stable with 1500 Byte packets, with no tests generating error messages of any kind. Every test, with flow durations of up to 2 hours, remained stable and was terminated gracefully at the predetermined time, using all kernel versions of 2.6.19 or later. Having a stable test bed has allowed preliminary tests of DCCP throughput performance, as outlined below.

6.1 Back-to-back tests

With any two systems CCID2 can attain the maximum possible data rate of 940 Mbit/s, which is line-rate over Gigabit Ethernet and stable for the duration of the test. This result is illustrated in Figure 1(a), with Figure 1(b) showing the different behaviour of CCID3. With CCID3 there is an

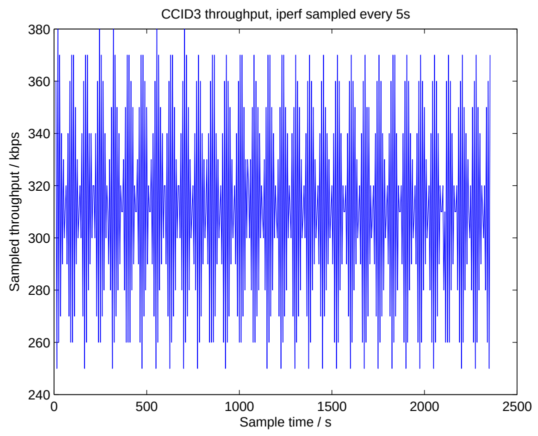


(a) CCID2



(b) CCID3

Figure 1: CCID comparison



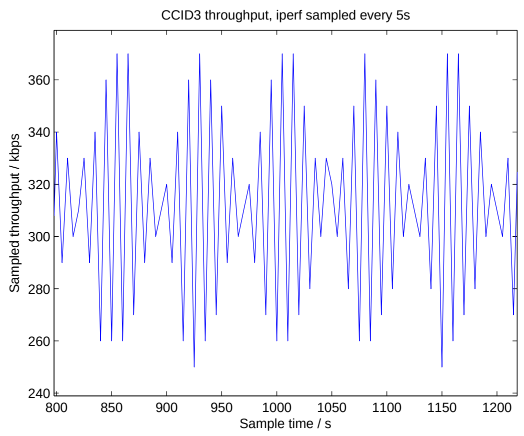


Figure 2: Expanded view of initial CCID3 throughput variation

initial period with an average rate of 300 Kbit/s, with the regular rate variation detailed in Figure 2. After a number of packets (around 65,500) the rate jumped to line-rate and remained steady, as seen in Figure 1(b). This is strange behaviour, with the number of packets being indicative with a 16-bit overflow perhaps, but there has been a lot of patches produced for CCID3 recently which have not yet made it into the stable Linux tree. Using a development tree and patches from numerous authors changes the CCID3 behaviour completely. The most appropriate comment to make is that CCID3 is developing and the performace of current stable kernels is not indicative of what is beiachievedved by developers.

6.2 Tests over extended networks

Over a transatlantic connection, with end-hosts in Manchester and Chicago, using UDP and TCP we can achieve line-rate throughput. Although the back-to-back performance of DCCP between identical systems gave line-rate, over the 94 ms transatlantic lightpath only a steady 130 Mbit/s was attained.

The performance of DCCP seemed to be CPU limited at the sender, with one CPU showing an average of 98% load, compared to the load at line-rate back-to-back of 82%. Increased CPU load with increasing round-trip time can sometimes be observed with TCP flows but it is not immediately obvious that this should be the case with DCCP and it is curious that the effect seems so dramatic. The performance of DCCP needs to be investigated further over different distances and with different systems. CPU load profiles can hopefully yield further useful information about the performance of DCCP.

7. Developing a new CCID

VLBI has a clear requirement to move constant bit-rate data and can tolerate high-levels of packet loss, making UDP seem like the ideal transport protocol. Other applications have similar requirements, with streaming media and VoIP being examples of applications where constant bit-rate can be advantageous and packet loss is often tolerable. However, there is concern from network providers that UDP traffic could overwhelm other traffic and overload the network. Concerns and opinions have been voiced and mitigating options have been discussed at recent meetings such as the EXPReS & EVN-NREN meeting in Zaandan, NL and PFLDnet 2007 / IRTF workshop in Marina Del Rey, US, with input from Kees Neggers, SURFnet; Glen Turner, AARNET; Aaron Falk, IRTF Chair. One option that the authors support is to use DCCP in combination with a new CCID, initially given the name *SafeUDP*. The proposed CCID aims to address the concerns expresses about using plain UDP by implementing something “UDP like” but with network protection.

SafeUDP would use the DCCP ACK mechanism to detect congestion, following which the congestion would be evaluated: to ensure that congestion is not in the end-host and to determine whether the congestion is transient. This evaluation step is useful to remove the assumption that all losses are congestion events, which is a conservative assumption but in some circumstances often unnecessarily detrimental to performance. The application would be notified of the congestion through modified API calls, with *sendto* and *recv_from*, etc. having new return codes. The application can then take action, with the CCID dropping input from the application and informing the application that it has done so if no action is taken. This idea is being worked on with the long-term aim of a draft RFC.

8. Conclusions

The Linux implementation of DCCP is almost certainly the most mature implementation available. Once we had established a stable test bed we investigated the performance of DCCP using CCID2 and CCID3 with tests conducted primarily using the test program *DCCPmon*, a port of existing application *UDPmon*. Apart from minor troubles due to omissions or changes to the API, the port was relatively straight-forward.

We have seen that the back-to-back performance of DCCP using CCID2 is good, achieving line-rate for extended (multiple hour) back-to-back, memory-to-memory transfers. The throughput of CCID3 was generally lower though there is much current development with performance changing with every patch. Given the amount of patches being created by developers it is uncertain at what speed the CCID3 implementation in the stable kernel will develop.

Tests of CCID2 over extended networks have been quite limited to date, with early results showing that DCCP uses much more CPU time and achieves a lower rate over a transatlantic lightpath. A rate of 130 Mbit/s to compare with 940 Mbit/s back-to-back has been seen, with further work needed to fully assess DCCP performance over long-distances.

achieveive a stable test setup has not been trivial and there are some issues still to be resolved. We hope that our investigations of the issues with DCCP on our systems can help improve the implementation and make DCCP work “out-of-the” box on more systems. Working round the issues we encountered revealed a protocol implementation that we look forward to investigating more fully in the near future. Many applications can benefit from DCCP and we hope to extend the utility by considering the concerns of and working with network managers to build a new CCID.

References

[1] R. Hughes-Jones, *DCCPmon Home Page*. Available at :
http://www.hep.man.ac.uk/u/rich/Tools_Software/dccpmon.html

[2] National Laboratory for Applied Network Research, *NLANR/DAST : Iperf*. Available at :
http://dast.nlanr.net/Projects/Iperf/