Solaris™ 2.x - Tuning Your TCP/IP Stack and More

Last update: 10.04.2001 (*change log*)

Please check your location line carefully. If you don't see http://www.sean.de/Solaris/ in your location bar, you might want to check with the original site for the most up to date information.

Important Notice!

SUN managed to publish a Solaris Tunable Parameters Reference Manual, applying to Solaris 8, HW 01/01. You might want to check there for anything you miss here.

Table of contents

Appendices are separate documents. They are quoted from within the text, but you might be interested in them when downloading the current document. If you say "print" for this document, the appendices will **not** be printed. You have to download and print them separately.

1. Introduction

Use at your own risk!

If your system behaves erratically after applying some tweaks, please don't blame me. Remember to have a backup handy before starting to tune. Always make backup copies of the files you are changing. I tried carefully to assemble the information you are seeing here, aimed at improved system performance. As usual, there are no guarantees that what worked for me will work for you. Please don't take my recommendation at heart: They are starting points, not absolutes. Always read my reasoning, don't use them blindly.

Before you start, you ought to grab a copy of the *TCP state transition diagram* as specified in RFC 793 on page 23. The drawback is the missing error correction supplied by later RFCs. There is an easier way to obtain blowup printouts to staple to your office walls. Grab a copy of the PostScript file pocket guide, page 2 accompanying Stevens' TCP/IP Illustrated Volume 1 [4]. Or simply open the book at figure 18.12.

Please share your knowledge

I try to assemble this page and related material for everybody interested in gaining more from her or his system. If you have an item I didn't cover, but which you deem worthwhile, please write to me. A few dozen or so regular readers of this page will thank you for it. I am only human, thus if you stumble over an error, misconception, or blatant nonsense, please have me correct it. In the past, there were quite a few mistakes.

The set of documents may look a trifle colorful, or just odd, if your browser supports cascading stylesheets. Care was taken to select the formatting tags in a way that the printed output still resembles the intentions of the author, and that the set of documents is still viewable with browser like Mosaic or Lynx. Stylesheets were used as an optical enhancement. Most notable is the different color of interior and external links. Interior links are shown in greenish colors, and will be rendered within the same frame. External links on the other hand are shown in bluish colors, and all will be shown in the same new frame. If you leave it open, a new external link will be shown within the same window. Literature references within the text are often interior links, pointing to the literature section, where the external links are located.

1.1 History

This page and the related work have a long history in gathering. I started out peeking wide eyed over the shoulders of two people from a search engine provider when they were installing the German server of a customer of my former employer. My only alternative resource of tuning information was the brilliant book TCP/IP Illustrated 1 [4] by Stevens. I started gathering all information about tuning I was able to get my hands upon. The cumulation of these you are experiencing on these pages.

1.2 Quick intro into ndd

Solaris allows you to tune, tweak, set and reset various parameters related to the TCP/IP stack *while the system is running*. Back in the SunOS 4.x days, one had to change various C files in the kernel source tree, generate a new kernel, reboot the machine and try out the changes. The Solaris feature of changing the important parameters on the fly is very convenient.

Many of the parameters I mention in the rest of the document you are reading are time intervals. All intervals are measured in **milliseconds**. Other parameters are usually bytecounts, but a few times different units of measurements are used and documented. A few items appear totally unrelated to TCP/IP, but due to the lack of a better framework, they materialized on this page.

Most tunings can be achieved using the program `ndd`. Any user may execute this program to read the current settings, depending on the readability of the respective device files. But only the super user is allowed to execute `ndd -set` to change values. This makes sense considering the sensitive parameters you are tuning. Details on the use of `ndd` can be

obtained from the respective manual page.

`ndd` will become your friend, as it is the major tool to tweak most of the parameters described in this document. Therefore you better make yourself familiar with it. A quick overview will be given in this section, too. `ndd` is not limited to tweaking TCP/IP related parameters. Many other devices, which have a device file underneath `/dev` and a kernel module can be configured with the help of `ndd`. For instance, any networking driver which supports the *Data Link Provider Interface (DLPI)* can be configured.

The parameters supplied to `ndd` are symbolic keys indexing either a single usually numerically value, or a table. Please note that the keys usually (but not always) start out with the module or device name. For instance, changing values of the IP driver, you have to use the device file `/dev/ip` and all parameters start out with `ip_`. The question mark is the most notable exception to this rule.

1.2.1 Interactive mode

The interactive mode allows you to inspect and modify a device, driver or module interactively. In order to inspect the available keyword names associated with a parameter, just type the question mark. The next item will explain about the output format of the parameter list.

```
# ndd /dev/tcp
name to get/set ? tcp_slow_start_initial
value ?
length ?
2
name to get/set ? ^D
```

The example above queries the TCP driver for the value of the slow start feature in an interactive fashion. The typed input is shown boldface.

1.2.2 Show all available parameters

If you are interested in the parameters you can tweak for a given module, query for the question mark. This special parameter name is part of all `ndd` configurable material. It tells the names of all parameters available - including itself - and the access mode of the parameter.

```
# ndd /dev/icmp \?
?                               (read only)
icmp_wroff_extra                (read and write)
icmp_def_ttl                    (read and write)
icmp_bsd_compat                 (read and write)
icmp_xmit_hiwat                 (read and write)
icmp_xmit_lowat                 (read and write)
icmp_recv_hiwat                 (read and write)
icmp_max_buf                    (read and write)
```

Please mind that you have to escape the question mark with a backslash from the shell, if you are querying in the non-interactive fashion as shown above.

1.2.3 Query the value of one or more parameters (read access)

At the command line, you often need to check on settings of your TCP/IP stack or other parameters. By supplying the parameter name, you can examine the current setting. It is permissible to mention several parameters to check on at once.

```
# ndd /dev/udp udp_smallest_anon_port
32768
# ndd /dev/hme link_status link_speed link_mode
1

1

1
```

The first example checks on the smallest anonymous port UDP may use when sending a PDU. Please refer to the appropriate section later in this document on the recommended settings for this parameter.

The second example checks the three important link report values of a 100 Mbit ethernet interface. The results are separated by an empty line, because some parameters may refer to tabular values instead of a single number.

1.2.4 Modify the value of one parameter (write access)

This mode of interaction with `ndd` will frequently be found in scripts or when changing value at the command line in a non-interactive fashion. Please note that you may only set one value at a time. The scripts section below contains examples

in how to make changes permanent using a startup script.

```
# ndd -set /dev/ip ip_forwarding 0
```

The example will stop the forwarding of IP PDUs, even if more than one non-local interface is active and up. Of course, you can only change parameters which are marked for both, reading and writing.

1.2.5 Further remarks

Andres Kroonmaa kindly supplied a nifty script to check all existing values for a network component (tcp, udp, ip, icmp, etc.). Usually I do the same thing using a small Perl script.

1.3 How to read this document

This document is separated into several chapters with little inter-relation. It is still advisable to loosely follow the order outlined in the table of contents.

The first chapter entirely focusses on the TCP connection queues. It is quite long for such small topic, but it is also meant to introduce you into my style of writing. The next chapter deals with TCP retransmission related parameters that you can adjust to your needs. The chapter is more concise. One chapter on deals with path MTU discovery, as there used to be problems with older versions of Solaris. Recent versions usually do not need any adjustments.

The fifth chapter is a kind of catch-all. Some TCP, some UDP and some IP related parameters are explained (forwarding, port ranges, timers), and a quick detour into bug 1226653 explains that some versions were capable of sending packages larger than the MTU. The following chapter in depth deals with windows, buffers and related issues.

Chapter seven detours from the `ndd` interface, and focusses on variables you can set in your `/etc/system` file, as some things can only be thus managed. Another part of that chapter deals with the `hme` interface and appropriate tunables. The chapter may be split in future, and parts of it are already found in the appendices.

The chapter dealing with patches, an important topic with any OS, just points you to various sources, and only mentions some essential things for older versions of Solaris.

Literature exists in abundance. The literature sections is more a lose collection of links and some books that I consider essential when working with TCP/IP, not limited to Solaris. The RFC sections is kind of hard to keep up-to-date, but then, I reckon you know how to read the rfc-index file.

The final chapters quickly glance at new or at one time new versions of Solaris - time makes them obsolete. The chapter is there for historical reason, more or less. The scripts sections deals with the `nettune` script used by YaSSP. It finishes with some TODO material.

2. TCP connection initiation

This section is dedicated exclusively to the various queues and tunable variable(s) used during connection instantiation. The socket API maintains some control over the queues. But in order to tune anything, you have to understand how `listen` and `accept` interact with the queues. For details, see the various Stevens books mentioned in the literature section.

When the server calls `listen`, the kernel moves the socket from the TCP state `CLOSED` into the state `LISTEN`, thus doing a passive open. All TCP servers work like this. Also, the kernel creates and initializes various data structures, among them the socket buffers and two queues:

incomplete connection queue

> This queue contains an entry for every `SYN` that has arrived. BSD sources assign `so_q0len` entries to this queue. The server sends off the `ACK` of the client's `SYN` and the server side `SYN`. The connection get queued and the kernel now awaits the completion of the TCP three way handshake to open a connection. The socket is in the `SYN_RCVD` state. On the reception of the client's `ACK` to the server's `SYN`, the connection stays one *round trip time (RTT)* in this queue before the kernel moves the entry into the

completed connection queue

> This queue contains an entry for each connection for which the three way handshake is completed. The socket is in the `ESTABLISHED` state. Each call to `accept()` removes the front entry of the queue. If there are no entries in the queue, the call to `accept` usually blocks. BSD source assign a length of `so_qlen` to this queue.

Both queues are limited regarding their number of entries. By calling `listen()`, the server is allowed to specify the size of the second queue for completed connections. If the server is for whatever reason unable to remove entries from the *completed connection queue*, the kernel is not supposed to queue any more connections. A timeout is associated with each received and queued `SYN` segment. If the server never receives an acknowledgment for a queued `SYN` segment, TCP state `SYN_RCVD`, the time will run out and the connection thrown away. The timeout is an important resistance against `SYN`

flood attacks.

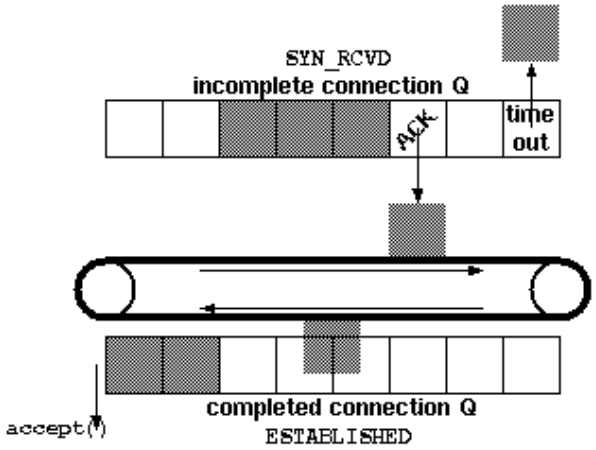

Figure 1: Queues maintained for listening sockets.

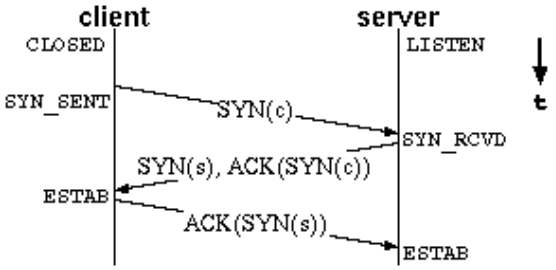

Figure 2: TCP three way handshake, connection initiation.

Historically, the argument to the listen function specified the maximum number of entries for the sum of both queues. Many BSD derived implementations multiply the argument with a *fudge factor* of 3/2. Solaris <= 2.5.1 do not use the fudge factor, but adds 1, while Solaris 2.6 does use the fudge factor, though with a slightly different rounding mechanism than the one BSD uses. With a backlog argument of 14, Solaris 2.5.1 servers can queue 15 connections. Solaris 2.6 server can queue 22 connections.

Stevens shows that the incomplete connection queue does need **more** entries for busy servers than the completed connection queue. The only reason for specifying a large backlog value is to enable the incomplete connection queue to grow as `SYN` arrive from clients. Stevens shows that moderately busy webserver has an empty *completed connection queue* during 99 % of the time, but the *incomplete connection queue* needed 15 or less entries in 98 % of the time! Just try to imagine what this would mean for a really busy webcache like Squid.

Data for an established connection which arrives before the connection is `accept()`ed, should be stored into the socket buffer. If the queues are full when a `SYN` arrived, it is dropped in the hope that the client will resend it, hopefully finding room in the queues then.

According to Cockroft [2], there was only one listen queue for unpatched Solari <= 2.5.1. Solari >= 2.6 or an applied TCP patch 103582-12 or above splits the single queue in the two shown in figure 1. The system administrator is allowed to tweak and tune the various maxima of the queue or queues with Solaris. Depending on whether there are one or two queues, there are *different* sets of tweakable parameters.

The old semantics contained just one tunable parameter **tcp_conn_req_max** which specified the maximum argument for the `listen()`. The patched versions and Solaris 2.6 replaced this parameter with the two new parameters **tcp_conn_req_max_q0** and **tcp_conn_req_max_q**. A SunWorld article on 2.6 by Adrian Cockroft tells the following about the new parameters:

> ***tcp_conn_req_max*** *[is] replaced. This value is well-known as it normally needs to be increased for Web servers in older releases of Solaris 2. It no longer exists in Solaris 2.6, and patch 103582-12 adds this feature to Solaris 2.5.1. The change is part of a fix that prevents denial of service from `SYN` flood attacks. There are now two separate queues of partially complete connections instead of one.*
>
> ***tcp_conn_req_max_q0*** *is the maximum number of connections with handshake incomplete. A `SYN` flood attack could only affect this queue, and a special algorithm makes sure that valid connections can still get through.*
>
> ***tcp_conn_req_max_q*** *is the maximum number of completed connections waiting to return from an accept call as soon as the right process gets some CPU time.*

In other words, the first specifies the size of the *incomplete connection queue* while the second parameters assigns the maximum length of the *completed connection queue*. All **three** parameters are covered below.

You can determine if you need to tweak this set of parameters by watching the output of `netstat -sP tcp`. Look for the value of `tcpListenDrop`, if available on your version of Solaris. Older versions don't have this counter. Any value showing up might indicate something wrong with your server, but then, killing a busy server (like squid) shuts down its listening socket, and might increase this counter (and others). If you get many drops, you might need to increase the appropriate parameter. Since connections can also be dropped, because `listen()` specifies a too small argument, you have to be careful interpreting the counter value. On old versions, a `SYN` flood attack might also increase this counter.

Newer or patched versions of Solaris, with both queues available, will also have the additional counters `tcpListenDropQ0` and `tcpHalfOpenDrop`. Now the original counter `tcpListenDrop` counts only connections dropped from the *completed connection queue*, and the counter ending in `Q0` the drops from the *incomplete connection*

queue. Killing a busy server application might increase either or both counters. If the `tcpHalfOpenDrop` shows up values, your server was likely to be the victim of a `SYN` flood. The counter is only incremented for dropping noxious connection attempts. I have no idea, if those will also show up in the `Q0` counter, too.

tcp_conn_req_max

default 8 (max. 32), since 2.5 32 (max. 1024), recommended 128 <= x <= 1024
since 2.6 **or** 2.5.1 with patches 103630-09 and 103582-12 or above applied:
see **tcp_conn_req_max_q** and **tcp_conn_req_max_q0**

The current parameter describes the maximum number of pending connection requests queued for a listening endpoint in the *completed connection queue*. The queue can only save the specified finite number of requests. If a queue overflows, nothing is sent back. The client will time out and (hopefully) retransmit.

The size of the *completed connection queue* does not influence the maximum number of simultaneous established connections after they were `accepted` **nor** does it have any influence on the maximum number of clients a server can serve. With Solaris, the maximum number of file descriptors is the limiting factor for simultaneous connections, which just happened to coincide with the maximum backlog queue size.

From the viewpoint of TCP those connections placed in the *completed connection queue* are in the TCP state `ESTABLISHED`, even though the application has not reaped the connection with a call to `accept`. That is the number limited by the size of the queue, which you tune with this parameter. If the application, for some reason, does not release entries from the queue by calling `accept`, the queue might overflow, and the connection is dropped. The client's TCP will hopefully retransmit, and might find a place in the queue.

Solaris offers the possibility to place connections into the backlog queue as soon as the first `SYN` arrives, called *eager listening*. The three way handshake will be completed as soon as the application `accept()`s the connection. The use of *eager listening* is not recommended for production systems.

Solari < 2.5 have a maximum queue length of 32 pending connections. The length of the *completed connection queue* can also be used to decrease the load on an overloaded server: If the queue is completely filled, remote clients will be denied further connections. Sometimes this will lead to a *connection timed out* error message.

Naively, I assumed that a very huge length might lead to a long service time on a loaded server. Stevens showed that the *incomplete connection queue* needs much more attention than the *completed connection queue*. But with **tcp_conn_req_max** you have no option to tweak that particular length.

Earlier versions of this document suggested to tune **tcp_conn_req_max** with regards to the values of **rlim_fd_max** and **rlim_fd_cur**, but the interdependencies are more complex than any rule of thumb. You have to find your own ideal. When a connection is still in the queue, only the queue length limits the number of entries. Connections taken from the queue are put into a file descriptor each.

> There is a trick to overcome the hardcoded limit of 1024 with a patch. SunSolve shows this trick in connection with `SYN` flood attacks. A greatly increased listen backlog queue may offer some small increased protection against this vulnerability. On this topic also look at the **tcp_ip_abort_cinterval** parameter. Better, use the mentioned TCP patches, and increase the q0 length.
>
> ```
> echo "tcp_param_arr+14/W 0t10240" | adb -kw /dev/ksyms /dev/mem
> ```
>
> This patch is only effective on the currently active kernel, limiting its extend to the next boot. Usually you want to append the line above on the startup script `/etc/init.d/inetinit`. The shown patch increases *hard limit* of the listen backlog queue to 10240. *Only after* applying this patch you may use values above 1024 for the **tcp_conn_req_max** parameter.

A further warning: Changes to the value of **tcp_conn_req_max** parameter in a running system **will not take effect** until each listening application is restarted. The backlog queue length is evaluated whenever an application calls `listen(3N)`, usually once during startup. Sending a HUP signal may or may not work; personally I prefer to TERM the application and restart them manually or, even better, use a startup script.

tcp_conn_req_max_q0

since 2.5.1 with patches 103630-09 and 103582-12 or above applied: default 1024;
since 2.6: default 1024, recommended 1024 <= x <= 10240

After installing the mentioned TCP patches, alternatively after installing Solaris 2.6, the parameter **tcp_conn_req_max** is no longer available. In its stead the new parameters **tcp_conn_req_max_q** and **tcp_conn_req_max_q0** emerged. **tcp_conn_req_max_q0** is the maximum number of connections with handshake incomplete, basically the length of the *incomplete connection queue*.

In other words, the connections in this queue are just being instantiated. A `SYN` was just received from the client, thus the connection is in the TCP `SYN_RCVD` state. The connection cannot be `accept()`ed until the handshake is complete, even if the *eager listening* is active.

To protect against *SYN flooding*, you can increase this parameter. Also refer to the parameter **tcp_conn_req_max_q** above. I believe that changes won't take effect unless the applications are restarted.

tcp_conn_req_max_q

since 2.5.1 with patches 103630-09 and 103582-12 or above applied: default 128;
since 2.6: default 128, recommended 128 <= x <= **tcp_conn_req_max_q0**

After installing the mentioned TCP patches, alternatively after installing Solaris 2.6, the parameter **tcp_conn_req_max** is no longer available. In its stead the new parameters **tcp_conn_req_max_q** and **tcp_conn_req_max_q0** emerged. **tcp_conn_req_max_q** is the length of the *completed connection queue.*

In other words, connections in this queue of length **tcp_conn_req_max_q** have completed the three way handshake of a TCP open. The connection is in the state `ESTABLISHED`. Connections in this queue have not been `accept()`ed by the server process (yet).

Also refer to the parameter **tcp_conn_req_max_q0**. Remember that changes won't take effect unless the applications are restarted.

tcp_conn_req_min

Since 2.6: default 1, recommended: don't touch

This parameter specifies the minimum number of available connections in the *completed connection queue* for `select()` or `poll()` to return "readable" for a listening (server) socket descriptor.

Programmers should note that Stevens [7] describes a timing problem, if the connection is `RST` between the `select()` or `poll()` call and the subsequent `accept()` call. If the listening socket is blocking, the default for sockets, it will block in `accept()` until a valid connection is received. While this seems no tragedy with a webserver or cache receiving several connection requests per second, the application is not free to do other things in the meantime, which might constitute a problem.

3. Retransmission related parameters

The *retransmission timeout* values used by Solaris are way too aggressive for wide area networks, although they can be considered appropriate for local area networks. SUN thus did not follow the suggestions mentioned in RFC 1122. Newer releases of the Solaris kernel are correcting the values in question:

> *The recommended upper and lower bounds on the RTO are known to be inadequate on large internets. The lower bound SHOULD be measured in fractions of a second (to accommodate high speed LANs) and the upper bound should be 2*MSL, i.e., 240 seconds.*

Besides the *retransmit timeout (RTO)* value two further parameters R1 and R2 may be of interest. These don't seem to be tunable via any Solaris' offered interface that I know of.

> *The value of R1 SHOULD correspond to at least 3 retransmissions, at the current RTO. The value of R2 SHOULD correspond to at least 100 seconds.*
>
> *[...]*
>
> *However, the values of R1 and R2 may be different for SYN and data segments. In particular, R2 for a SYN segment MUST be set large enough to provide retransmission of the segment for at least 3 minutes. The application can close the connection (i.e., give up on the open attempt) sooner, of course.*

Great many internet servers which are running Solaris do retransmit segments unnecessarily often. The current condition of European networks indicate that a connection to the US may take up to 2 seconds. All parameters mentioned in the first part of this section relate to each other!

As a starter take this little example. Consider a picture, size 1440 byte, LZW compressed, which is to be transferred over a serial linkup with 14400 bps and using a MTU of 1500. In the ideal case only one PDU gets transmitted. The `ACK` segment can only be sent *after* the complete PDU is received. The transmission takes about 1 second. These values seem low, but they are meant as 'food for thought'. Now consider something going awry...

Solaris 2.5.1 is behaving strange, if the initial `SYN` segment from the host doing the active open is lost. The initial `SYN` gets retransmitted only after a period of 4 * **tcp_rexmit_interval_initial** plus a constant C. The time is 12 seconds with the default settings. More information is being prepared on the retransmission test page.

The initial lost `SYN` may or may not be of importance in your environment. For instance, if you are connected via ATM SVCs, the initial PDU might initiate a logical connection (ATM works point to point) in less than 0.3 seconds, but will still be lost in the process. It is rather annoying for a user of 2.5.1 to wait 12 seconds until something happens.

tcp_rexmit_interval_initial

default 500, since 2.5.1 3000, recommended >= 2000 (500 for special purposes)

This interval is waited before the last data sent is retransmitted due to a missing acknowledgment. Mind that this interval is used only for the *first* retransmission. The more international your server is, the larger you should chose this interval.

Special laboratory environments working in LAN-only environments might be better off with 500 ms or even less. If you are doing measurements involving TCP (which is almost always a bad idea), you should consider lowering this parameter.

> Why do I consider TCP measurements a bad idea? If ad-hoc approaches are used, or there is no deeper knowledge of the mechanics of TCP, you are bound to arrive at wrong conclusions. Unless there are TCP dumps to document that indeed what you expect is actually happening, results may lead to wrong conclusions. If done properly, there is nothing wrong with TCP measurements. The same rules apply, if you are measuring protocols on top of TCP.
>
> There are lots of knobs and dials to be fiddled with - all of which need to be documented along with the results. Scientific experiments need to be repeatable by others in order to verify your findings.

tcp_rexmit_interval_min

default 200, recommended >= 1000 (200 for special purposes)
Since 8: default 400

After the initial retransmission further retransmissions will start after the **tcp_rexmit_interval_min** interval. BSD usually specifies 1500 milliseconds. This interval should be tuned to the value of **tcp_rexmit_interval_initial**, e.g. some value between 50 % up to 200 %. The parameter has no effect on retransmissions during an *active open*, see my accompanying document on retransmissions.

> The **tcp_rexmit_interval_min** doesn't display any influence on connection establishment with Solaris 2.5.1. It does with 2.6, though. The influence on regular data retransmissions, or FIN retransmissions I have yet to research.

tcp_ip_abort_interval

default 120000, since 2.5 480000, recommended 600000

This interval specifies how long *retransmissions* for a connection in the `ESTABLISHED` state should be tried before a `RESET` segment is sent. BSD systems default to 9 minutes.

tcp_ip_abort_cinterval

default 240000, since 2.5 180000, recommended ?

This interval specifies how long retransmissions for a remote host are repeated until the `RESET` segment is sent. The difference to the **tcp_ip_abort_interval** parameter is that this connection is about to be established - it has not yet reached the state `ESTABLISHED`. This value is interesting considering `SYN` flood attacks on your server. Proxy server are doubly handicapped because of their Janus behavior (like a server towards the downstream cache, like a client towards the upstream server).

According to Stevens this interval is connected to the *active open*, e.g. the `connect(3N)` call. But according to SunSolve the interval has an impetus on **both** directions. A remote client can refuse to acknowledge an opening connection up to this interval. After the interval a `RESET` is sent. The other way around works out, too. If the three-way handshake to open a connection is not finished within this interval, the `RESET` Segment will be sent. This can only happen, if the final `ACK` went astray, which is a difficult test case to simulate.

To improve your `SYN` flood resistance, SUN suggests to use an interval as small as 10000 milliseconds. This value has only been tested for the "fast" networks of SUN. The more international your connection is, the slower it will be, and the more time you should grant in this interval. Proxy server should never lower this value (and should let Squid terminate the connection). Webservers are usually not affected, as they seldom actively open connections beyond the LAN.

tcp_rexmit_interval_max

default 60000, RFC 1122 recommends 240000 (2MSL), recommended 1...2 * **tcp_close_wait_interval** or **tcp_time_wait_interval**
Since 2.6: default 240000
Since 8: default 60000

All previously mentioned retransmissions related interval use an *exponential backoff* algorithm. The wait interval between two consecutive retransmissions for the same PDU is doubled starting with the minimum.

The **tcp_rexmit_interval_max** interval specifies the maximum wait interval between two retransmissions. If changing this value, you should also give the abort interval an inspection. The maximum wait interval should only be reached shortly before the abort interval timer expires. Additionally, you should coordinate your interval with the value of **tcp_close_wait_interval** or **tcp_time_wait_interval**.

tcp_deferred_ack_interval

default 50, BSD 200, recommended 200 (regular), 50 (benchmarking), or 500 (WAN server)
Since 8: default 100

This parameter specifies the timeout before sending a *delayed ACK*. The value should **not be increased above 500,** as required by RFC 1122. This value is of great interest for interactive services. A small number will increase the "responsiveness" of a remote service (telnet, X11), while a larger value can decrease the number of segments exchanged.

The parameter might also interest to HTTP servers which transmit small amounts of data after a very short retrieval time. With a heavy-duty servers or in laboratory banging environment, you might encounter service times answering a request which are well above 50 ms. An increase to 500 might lead to less PDUs transferred over the network, because TCP is able to merge the ACK with data. Increases beyond 500 **should not** be even considered.

SUN claims that Solaris recognizes the initial data phase of a connection. An initial ACK (not SYN) is *not* delayed. As opposed to the simplistic approach mentioned in the SUN paper, a request for a webservice (both, server or proxy) which does not fit into a single PDU can be transmitted faster. Also check the **tcp_slow_start_initial** Parameter.

> The **tcp_deferred_ack_interval** also seems to be used to distinguish full-sized segments between interactive traffic and bulk data transfer. If a sender uses MSS sized segments, but sends each segment further apart than approximately 0.9 times the interval, the traffic will be rated interactive, and thus every segment seems to get ACKed.

tcp_deferred_acks_max

Since 2.6: default 8, recommended ?, maximum 16

This parameter features the maximum number of segments received after which an ACK just has to be sent. Previously I thought this parameter solely related to interactive data transfer, but I was mistaken. This parameter specifies the number of outstanding ACKs. You can give it a look when tuning for high speed traffic and bulk transfer, but the parameter is controversial. For instance, unless you employ selective acknowledgments (SACK) like Solaris 7, you can only ACK the number of segments correctly received. With the parameter at a larger value, statistically the amount of data to retransmit is larger.

Good values for retransmission tuning don't beam into existence from a white source. Rather you should carefully plan an experiment to get decent values. Intervals from another site **can not** be carried over to another Solaris system without change. But they might give you an idea where to start when choosing your own values.

The next part looks at a few parameters having to do with retransmissions, as well.

tcp_slow_start_initial

Since 2.5.1 with patch 103582-15 applied: default 1
Since 2.6: default 1, recommended 2 or 4 for servers
Since 8: default 4, no recommendations

This parameter provides the slow-start bug discovered in BSD and Windows TCP/IP implementations for Solaris. More information on the topic can be found on the servers of SUN and in Stevens [6]. To summarize the effect, a server starts sending two PDUs at once without waiting for an ACK due to wrong ACK counts. The ACK from connection initiation being counted as data ACK - compare with figure 2. Network congestion avoidance algorithms are being undermined. The slow start algorithm does not allow the buggy behavior, compare with RFC 2001.

Setting the parameter to 2 allows a Solaris machine to behave like it has the slow start bug, too. Well, IETF is said to make amends to the slow start algorithm, and the bug is now actively turned into a feature. SUN also warns:

> *It's still conceivable, although rare, that on a configuration that supports many clients on very slow-links, the change might induce more network congestions. Therefore the change of* ***tcp_slow_start_initial*** *should be made with caution.*
>
> *[...]*
>
> *Future Solaris releases are likely to default to 2.*

You can also gain performance, if many of your clients are running old BSD or derived TCP/IP stacks (like MS). I expect new BSD OS releases not to figure this bug, but then I am not familiar with the BSD OS family. A reader of this page told me about cutting the latency of his server in half, just by using the value of 2.

If you want to know more about this feature and its behavior, you can have a look at some experiments I have conducted concerning that particular feature. The summary is that I agree with the reader: A BSDish client like Windows definitely profits from using a value of 2.

tcp_slow_start_after_idle

Since 2.6: default 2, no recommendations
Since 8: default 4, no recommendations

I reckon that this parameter deals with the slow start for an already established connection which was idle for some time (however the term idle is defined here).

tcp_dupack_fast_retransmit

default 3, no recommendations

Something to do with the number of duplicates ACKs. If we do fast retransmit and fast recovery algorithms, this many ACKs must be retransmitted until we assume that a segment has really been lost. A simple reordering of segments usually causes no more than two duplicate ACKs.

There are a couple of parameters which require some elementary familiarity with RFC 2001, which covers TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, as well as *ssthresh* and *cwnd*.

tcp_rtt_updates

default 0, BSD 16, recommended: (see text)
Since 8: 20, no recommendations

This parameter controls when things like *rtt_sa* (the smoothed RTT), *rtt_sd* (the smoothed mean deviation), and *ssthresh* (the slow start threshold) are cached in the routing table. By default, Solaris does not cache any of the parameters. It is claimed that you can set it to a value you like, but to be the same as BSD, use 16.

The value to this parameter is the number of RTT samples that had to be sampled, so that an accurate enough value can be stored in the routing table. If you chose to use this feature, use a value of 16 or above. Using 16 allows the smoothed RTT filter to converge within 5 % of the correct value, compare Stevens [4], chapter 21.9.

ip_ire_cleanup_interval

default 30000, no recommendatations
Since 8: the parameter has a new name, just which one?

The parameters may do more than described here. If a routing table entry is not directly connected and not being used, the cache for things like *rtt_sa*, *rtt_sd* and *ssthresh* associated with the entry will be flushed after 30 seconds. The parameter **tcp_rtt_updates** must be greater than zero to enable the cache.

I could imagine that external helper programs invoked by MRTG on a regular basis connecting to a far-away host might benefit from increasing this value slightly above the invocation interval.

4. path MTU discovery

Whenever a connection is about to be established, the three-way handshake open negotiation, the segment size used will be set to the minimum of (a) the smallest MTU of an outgoing interface, and (b) from MSS announced by the peer. If the remote peer does not announce a MSS, usually the value 536 will be assumed. If path MTU discovery is active, all outgoing PDUs have the IP option DF (*don't fragment*) set.

If the ICMP error message *fragmentation needed* is received, a router on the way to the destination needed to fragment the PDU, but was not allowed to do so. Therefore the router discarded the PDU and did send back the ICMP error. Newer router implementations enclose the needed MSS in the error message. If the needed MSS is not included, the correct MSS must be determined by trial and error algorithm.

Due to the internet being a packet switching network, the route a PDU travels along a TCP virtual circuit may change with time. For this reason RFC 1191 recommends to rediscover the path MTU of an active connection after 10 minutes. Improvements of the route can only be noticed by repeated rediscoveries. Unfortunately, Solaris aggressively tries to rediscover the path MTU every 30 seconds. While this is o.k. for LAN environments, it is a grossly impolite behavior in WANs. Since routes may not change that often, aggressive repetitions of path MTU discoveries leads to unnecessary consumption of channel capacity and elongated service times.

Path MTU discovery is a far reaching and controversial topic when discussing it with local ISPs. Still, pMTU discovery is at the foundation of IPv6. The PSC tuning page argues pro path MTU discovery, especially if you maintain a high-speed or long-delay (e.g. satellite) link.

The recommendation I can give you is **not** to use the defaults of Solaris < 2.5. Please use path MTU discovery, but tune your system RFC conformant. You may alternatively want to switch off the path MTU discovery all together, though there are few situations where this is necessary.

I was made aware of the fact that in certain circumstances bridges connecting data link layers of differing MTU sizes defeat pMTU discovery. I have to put some more investigation into this matter. If a frame with maximum MTU size is to be transported into the network with the smaller MTU size, it is truncated silently. A bridge does not know anything about the upper protocol levels: A bridge neither fragments IP nor sends an ICMP error.

There may be work-arounds, and the **tcp_mss_def** is one of them. Setting all interfaces to the minimum shared MTU might help, at the cost of losing performance on the larger MTU network. Using what RFC 1122 calls an *IP gateway* is a possible, yet expensive solution.

ip_ire_pathmtu_interval

default 30000, recommended 600000
Since 2.5 600000, no recommendations

This timer determines the interval Solaris rediscovers the path MTU. An extremely large value will only evaluate the path MTU once at connection establishment.

ip_path_mtu_discovery

default 1, recommended 1

This parameter switches path MTU discovery on or off. If you enter a 0 here, Solaris will never try to set the DF bit in the IP option - unless your application explicitly requests it.

tcp_ignore_path_mtu

default 0, recommended 0

This is a *debug switch*! When activated, this switch will have the IP or TCP layer ignore all ICMP error messages *fragmentation needed*. By this, you will achieve the opposite of what you intended.

tcp_mss_def

default 536, recommended >= 536
Since 8: split into **tcp_mss_def_ipv4** and **tcp_mss_def_ipv6**

This parameter determines the default MSS (*maximum segment size*) for non-local destination. For path MTU discovery to work effectively, this value can be set to the MTU of the most-used outgoing interface descreased by 20 byte IP header and 20 byte TCP header - if and only if the value is bigger than 536.

tcp_mss_def_ipv4

Since 8: default 536

tcp_mss_def_ipv6

Since 8: default 1460

Solaris 8 supports IPv6. Since IPv6 uses different defaults for the maximum segment size, one has to distinguish between IPv4 and IPv6. The default for IPv6 is close to what is said for **tcp_mss_def**.

5. Further advice, hints and remarks

This section covers a variety of topics, starting with various TCP timers which do not relate to previously mentioned issues. The next subsection throws a quick glance at some erratic behavior. The final section looks at a variety of parameters which deal with the reservation of resources.

Additionally, I strongly suggest the use of a file `/etc/init.d/nettune` (always called *first script*) which changes the tunable parameters. `/etc/rcS.d/S31nettune` is a hardlink to this file. The script will be executed during bootup when the system is in *single user mode*. A killscript is not necessary. The section about startup scripts below reiterates this topic in greater depth.

5.1 Common TCP timers

The current subsection covers three important TCP timers. First I will have a look at the keepalive timer. The timer is rather controversial, and some Solari implement them incorrectly. The next parameter limits the *twice maximum segment lifetime (2MSL)* value, which is connected to the time a socket spends in the TCP state `TIME_WAIT`. The final entry looks at the time spend in the TCP state `FIN_WAIT_2`.

tcp_keepalive_interval

default 7200000, minimum 10000, recommended 10000 <= x <= oo

This value is one of the most controversial ones when talking with other people about appropriate values. The interval specified with this key must expire before a *keep-alive probe* can be sent. Keep-alive probes are described in the host requirements RFC 1122: If a host chooses to implement *keep-alive* probes, it **must** enable the application to switch them on or off for a connection, and keep-alive probes **must** be switched off by default.

Keep-alives can terminate a perfectly good connection (as far as TCP/IP is concerned), cost your money and use up transmission capacity (commonly called bandwidth, which is, actually, something completely different). Determining whether a peer is alive should be a task of the application and thus kept on the application layer. Only if you run into the danger of keeping a server in the `ESTABLISHED` state forever, and thus using up precious server resources, you should switch on keep-alive probes.

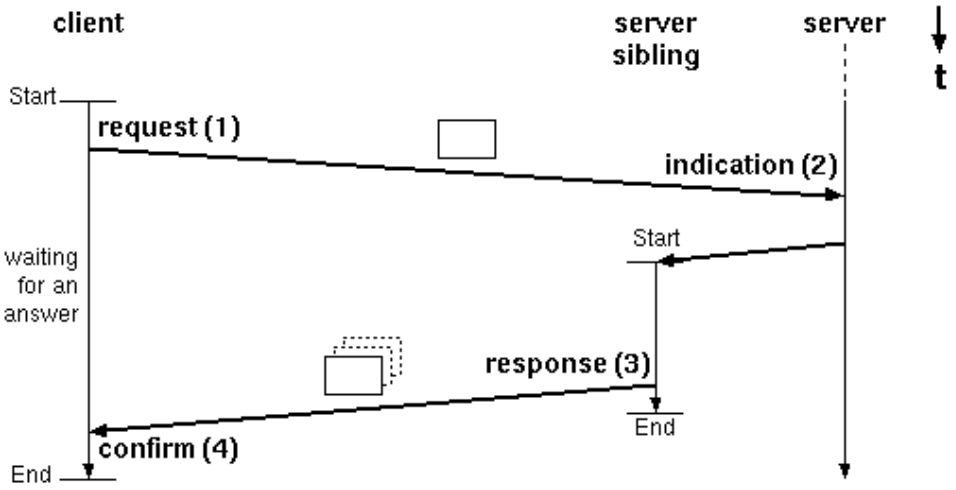

Figure 3: A typical handshake during a transaction.

Figure 3 shows the typical handshake during a HTTP connection. It is of no importance for the argumentation if the server is threaded, preforked or just plain forked. Webservers work transaction oriented as is shown in the following simplified description - **the numbers *do not relate* to the figure:**

1. The client (browser) initiates a connection (*active open*).
2. The client forwards its query (*request*).
3. The server (daemon) answers (*response*).
4. The server terminates the connection (*active close*).

Common implementations need to exchange **9..10 TCP segments per HTTP connection**. The *keep-alive* option as a HTTP/1.0 protocol and extensions can be regarded as a *hack*. Persistent connections are a different matter, and not shown here. Most people still use HTTP/1.0, especially the Squid users.

The *keep-alive* timer becomes significant for webservers, if in step 1 the client crashed or terminates without the server knowing about it. This condition can be forced sometimes by quickly pressing the stop button of netscape or the Logo of Mosaic. Thus the *keep-alive* probes do make sense for webservers. HTTP Proxies look like a server to the browser, but look like a client to the server they are querying. Due to their server like interface, the conditions for webservers are true for proxies, as well.

With an implementation of keep-alive probes **working correctly**, a very small value can make sense when trying to improve webservers. In this case you have to make sure that the probes stop after a finite time, if a peer does not answer. **Solari <= 2.5 have a bug** and send keep-alive probes forever. They seem to want to elicit some response, like a `RST` or some ICMP error message from an intermediate router, but never counted on the destination simply being down. Is this fixed with 2.5.1? Is there a patch available against this misbehavior? I don't know, maybe you can help me.

I am quite sure that this bug is fixed in 2.6 and that it is safe to use a small value like ten minutes. Squid users should synchronize their cache configuration accordingly. There are some Squid timeouts dealing with an idle connection.

tcp_close_wait_interval

default 240000 (according to RFC 1122, 2MSL), recommended 60000, possibly lower
Since 7: obsoleted parameter, use **tcp_time_wait_interval** instead
Since 8: no more access, use **tcp_time_wait_interval**

Even though the parameter key contains "close_wait" in its name, the value specifies the **TIME_WAIT** interval! In order to fix this kind of confusion, starting with Solaris 7, the parameter **tcp_close_wait_interval** was renamed to the correct name **tcp_time_wait_interval**. The old key **tcp_close_wait_interval** still exists for backward compatibility reasons. User of Solari *below* 7 must use the old name **tcp_close_wait_interval**
. Still, refer to **tcp_time_wait_interval** for an in-depth explaination.

tcp_time_wait_interval

Since 7: default 240000 (2MSL according to RFC 1122), recommended 60000, possibly lower

As Stevens repeatedly states in his books, the `TIME_WAIT` state is your friend. You should not desperately try to avoid it, rather try to understand it. The *maximum segment lifetime*(MSL) is the maximum interval a TCP segment may live in the net. Thus waiting twice this interval ensures that there are no leftover segments coming to haunt you. This is what the 2MSL is about. Afterwards it is safe to reuse the socket resource.

The parameter specifies the 2MSL according to the four minute limit specified in RFC 1122. With the knowledge about

current network topologies and the strategies to reserve ephemerical ports you should consider a shorter interval. The shorter the interval, the faster precious resources like ephemerical ports are available again.

A toplevel search engine implementor recommends a value of 1000 millisecond to its customers. Personally I believe this is too low for regular server. A loaded search engine is a different matter alltogether, but now you see where some people start tweaking their systems. I rather tend to use a multiple of the **tcp_rexmit_interval_initial** interval. The current value of **tcp_rexmit_interval_max** should also be considered in this case - even though retransmissions are unconnected to the 2MSL time. A good starting point might be the double RTT to a very remote system (e.g. Australia for European sites). Alternatively a German commercial provider of my acquaintance uses 30000, the smallest interval recommended by BSD.

tcp_fin_wait_2_flush_interval

BSD 675000, default 675000, recommended 67500 (one zero less)

This values seems to describe the (BSD) timer interval which prohibits a connection to stay in the `FIN_WAIT_2` state forever. `FIN_WAIT_2` is reached, if a connection closes actively. The `FIN` is acknowledged, but the `FIN` from the passive side didn't arrive yet - and maybe never will.

Usually webservers and proxies actively close connections - as long as you don't use persistent connection and even those are closed from time to time. Apart from that HTTP/1.0 compliant server and proxies close connections after each transaction. A crashed or misbehaving browser may cause a server to use up a precious resource for a long time.

You should consider decreasing this interval, if `netstat -f inet` shows many connections in the state `FIN_WAIT_2`. The timer is only used, if the connection is really *idle*. Mind that after a TCP half close a simplex data transmission is still available towards the actively closing end. TCP half closes are not yet supported by Squid, though many web servers do support them (certain HTTP drafts suggest an independent use of TCP connections). Nevertheless, as long as the client sends data after the server actively half closed an established connection the timer is not active.

Sometimes, a Squid running on Solaris (2.5.1) confuses the system utterly. A great number of connection to a varying degree are in `CLOSE_WAIT` for reasons beyond me. During this phase the proxy is virtually unreachable for HTTP requests though, obnoxiously, it still answers ICP requests. Although lowering the value for **tcp_close_wait_interval** is only fixing symptoms indirectly, not the cause, it may help overcoming those periods of erratic behavior faster than the default. The thing needed would be some means to influence the `CLOSE_WAIT` interval directly.

5.2 Erratic IPX behavior

I noticed that Solari < 2.6 behave erratically under some conditions, if the IPX ethernet MTU of 1500 is used. Maybe there is an error in the frame assembly algorithm. If you limit yourself to the IEEE 802.3 MTU of 1492 byte, the problem does not seem to appear. A sample startup script with link in `/etc/rc2.d` can be used to change the MTU of ethernet interfaces after their initialization. Remember to set the MTU for every virtual interface, too!

> Note, with a patched Solaris 2.5.1 or Solaris 2.6, the problem does not seem to appear. Limiting your MTU to non-standard might introduce problems with truncated PDUs in certain (admittedly very special) environments. Thus you may want to refrain from using the above mentioned script (always called *second script* in this document).
>
> Since I observed the erratic behavior only in a Solaris 2.5, I believe it has been fixed with patch 103169-10, or above. The error description reads "1226653 IP can send packets larger than MTU size to the driver."

5.3 Common IP parameters

The following parameters have little impact on performance, nevertheless I reckon them worth noting here. Please note that parameters starting with the **ip6** prefix apply to IPv6 while its twin with the **ip** applies to IPv4:

ip6_forwarding

Since 8: default 1, recommended 0 for pure server hosts or security

ip_forwarding

default 2, recommended 0 for pure server hosts or security
Since 8: default 1, recommended 0 for security reasons

If you intend to disable the routing abilities of your host all together, because you know you don't need them, you can set this switch to 0. The default value of 2 was only available in older versions of Solaris. It activates IP forwarding, if two or more real interfaces are up. The value of 1 in Solari < 8 activates IP forwarding regardless of the number of interfaces. With the possible exception of MBone routers and firewalling, you should leave routing to the dedicated routing hardware.

Starting with Solaris 8, the parameter set is split. You use **ip_forwarding** and **ip6_forwarding** to overall switch on forwarding of IPv4 and IPv6 PDU respectively between interfaces. The interfaces participating in forwarding can be activated separately, see **if:ip_forwarding**. Unless you host is acting as router, it is still recommended for security reasons to switch off any forwarding between interfaces.

if:ip_forwarding

Since 8: default 0, maximum 1, recommended 0

Please replace the *if* part of the parameter name with the appropriate interface available on your system, e.g. *hme0* or *hme0*. Look into the available `/dev/ip` parameters, if unsure what interfaces are known to the IP stack.

Starting with Solaris 8, a subset of interfaces participating in IP forwarding can be selected by setting the appropriate parameter to 1. You also need to set the **ip6_forwarding** and **ip_forwarding** parameter, if you want to forward IPv6 or IPv6 respectively.

For security reasons, and in many environments, forwarding is *not* recommended.

ip6_forward_src_routed

Since 8: default 1, recommended 0 for security reasons

ip_forward_src_routed

default 1, recommended 0 for security reasons

This parameter determines if IP datagrams can be forwarded which have the *source routing* option activated. The parameter has little meaning for performance but is rather of security relevance. Solaris may forward such datagrams, if the host route option is activated, bypassing certain security construct - possibly undermining your firewall. Thus you should disable it always, unless the host functions as a regular router (and no other services).

If you enabled IPv6 forwarding or IPv4 forwarding, the ***_forward_src_routed** parameters may relate to forwarding.

ip_forward_directed_broadcasts

default 1, recommended 0 for pure server hosts or security

This switch decides whether datagrams directed to any of your direct broadcast addresses can be forwarded as link-layer broadcasts. If the switch is on (default), such datagrams are forwarded. If set to zero, pings or other broadcasts to the broadcast address(es) of your installed interface(s) are silently discarded. The switch is recommended for any host, but can break "expected" behavior.

ip6_respond_to_echo_multicast

Since 8: default 1, recommended 0 for security reasons

ip_respond_to_echo_broadcast

default 1, recommended 0 for security reasons

If you don't want to respond to an ICMP echo request (usually generated by the `ping` program) to any of your IPv4 broadcast or IPv6 multicast addresses addresses, set the matching parameter to 0. On one hand, responding to broadcast pings is rumored to have caused panics, or at least partial network meltdowns. On the other hand, it is a valid behavior, and often used to determine the number of alive hosts on a particular network. If you are dead sure that neither you nor your network admin will need this feature, you can switch it off by using the value of 0.

If you do not want to respond to any IPv4-broadcast or IPv6-multicast probes for security reasons, it is recommended to set the matching parameter to 0.

ip_icmp_err_burst

Since 8: default 10, min 1, maximum 99999, see text

ip_icmp_err_interval

default 500, recommended: see text

Solaris IP only generates **ip_icmp_err_burst** ICMP error messages in any **ip_icmp_err_interval**, regardless of IPv4 or IPv6. In order to protect from *denial of service (DOS)* attacks, the parameters do not need to be changed. Some administrators may need a higher error generation rate, and thus may want to decrease the interval or increase the generated message.

In versions of Solaris prior to 8, **ip_icmp_err_interval** used to define the minimum time between two consecutive ICMP error responses - as if in older versions the (by then not existing) **ip_icmp_err_burst** parameter had a value of 1. The generated ICMP responses include the *time exceeded* message as evoked by the `traceroute` command. If your current setting here is above the RTT of a `traceroute` probe, usually the second probe you see will time out.

If you set **ip_icmp_err_burst** to exactly 0, `traceroute` will not give away your host as running Solaris. Also, you switched of the rate limitation of ICMP messages, and are thus open to DOS attacks. Of course, there are other ways to determine which TCP/IP implementation a networked host is running.

ip6_icmp_return_data_bytes

Since 8: default 64, minimum 8, maximum 65520, no recommendations

ip_icmp_return_data_bytes

default 64, minimum 8, maximum 65520, no recommendations

The parameters control the number of bytes returned by any ICMP error message generated on this Solaris host. The

default value 64 is sufficient for most cases. Some laboratory environments may want to temporarily increase the value in order to figure out problems with some network services.

ip6_send_redirects

Since 8: default 1, recommendation 0 for security reasons

ip_send_redirects

default 1, recommendation 0 for security reasons

These parameters control whether the IPv4 or IPv6 part of the IP stack send ICMP redirect messages. For security reasons, it is recommended to disable sending out such messages, unless your host is acting as router.

If you enabled IPv6 forwarding or IPv4 forwarding, the ***_send_redirects** parameters may relate to forwarding.

ip6_ignore_redirects

Since 8: default 0, recommendation 1 for security reasons

ip_ignore_redirect

default 0, recommendation 1 for security reasons

This flag control, if your routing table can be updated by ICMP redirect messages. Unless you run your host to act as router, it is recommended to disable this feature for security reasons. Otherwise, malicious external hosts may confuse your routing table.

If you enabled IPv6 forwarding or IPv4 forwarding, the ***_ignore_redirects** parameters may relate to forwarding.

ip_addrs_per_if

default 256, minimum 1, maximum 8192, no recommendations

This parameter limits the number of virtual interfaces you can declare per physical interface. Especially if you run Web Polygraph, you will need to increase the number of virtual interfaces available on your system.

ip6_strict_dst_multihoming

Since 8: default 0, recommended: see text

ip_strict_dst_multihoming

default 0, recommended: see text

According to RFC 1122, a host is said to be multihomed, if it has more than one IP address. Each IP address is assumed to be a logical interface. Different logical interfaces may map to the same physical interface. Physical interfaces may be connected to the same or different networks.

> The **strong end system model** aka strict multihoming requires a host not to accept datagrams on physical interfaces to which to logical one is not bound. Outgoing datagrams are restricted to the interface which corresponds with the source ip address.
>
> The **weak end system model** aka loose multihoming lets a host accept any of its ip addresses on any of its interfaces. Outgoing datagrams may be sent on any interface.

For security reasons, it is recommended to require strict multihoming, that is, setting the parameter to value 1. In certain circumstances, though, it may be necessary to disable strict multihoming, e.g. if the host is connected to a *virtual private networks (VPN)* or sometimes when acting as firewall.

For instance, I once maintained a setup, where a pair of related caching proxies were talking exclusively to each other via a crossover cable on one interface using private addresses while the other interface was connected to the public internet. In order to have them actually use the behind-the-scenes link, I had to manually set routes and disable strict multihoming.

5.4 TCP and UDP port related parameters

There are some parameters related to the ranges of ports associated with reserved access and non-privileged access. This section deals with the majority of useful parameters when selecting different than default port ranges.

udp_smallest_anon_port

tcp_smallest_anon_port

default 32768, recommended 8192

This value has the same size for UDP and TCP. Solaris allocates ephemerical ports above 32768. Busy servers or hosts using a large 2MSL, see **tcp_close_wait_interval**, may want to lower this limit to 8192. This yields more precious resources, especially for proxy servers.

A contra-indication may be servers and services running on well known ports above 8192. This parameter should be set very early during system bootup, especially before the portmapper is started.

The IANA port numbers document requires the assigned and/or private ports to start at 49152. For busy servers, severly limiting their ephemerical port supply in such a manner is not an option.

udp_largest_anon_port

default 65535, recommended: see text

This parameter has to be seen in combination with **udp_smallest_anon_port**. The `traceroute` program tries to reach a random UDP port above 32768 - or rather tries not to reach such a port - in order to provoke an ICMP error message from the host.

Paranoid system administrator may want to lower the value for this reason down to 32767, **after** the corresponding value for **udp_smallest_anon_port** has been lowered. On the other hand, datagram application protocols should be able to cope with foreign protocol datagrams.

If an ICP caching proxy or other UDP hyper-active applications are used, the lowering of this value can not be recommended. The respective TCP parameter **tcp_largest_anon_port** does not suffer this problem.

tcp_largest_anon_port

default 65535, no recommendations

The largest anonymous port for TCP should be the largest possible port number. There is no need to change this parameter.

udp_smallest_nonpriv_port

default 1024, no recommendations

tcp_smallest_nonpriv_port

default 1024, no recommendations

Privileged ports can only be bound to by the superuser. The smallest non-privileged port is the first port that a regular user can have his or her application to bind to.

tcp_extra_priv_ports_add

udp_extra_priv_ports_add

write-only action

tcp_extra_priv_ports_del

udp_extra_priv_ports_del

write-only action

tcp_extra_priv_ports

udp_extra_priv_ports

default (depends on active services)

The extra priviledged ports are those priviledged ports outside the scope of the reserved ports. Reserved port numbers are usually below 1024, see **tcp_smallest_nonpriv_port** for TCP and **tcp_smallest_nonpriv_port** for UDP, and require superuser privileges in order to bind to. For instance, if NFS is activated, the NFS server port 2049 is marked as privileged.

You can examine the extra privileged TCP port by looking at the read-only parameter **tcp_extra_priv_ports**. If you need to add an extra privileged port, use the **tcp_extra_priv_ports_add** with the port number as argument. If you need to remove an extra privileged port, use the **tcp_extra_priv_ports_del** action with the port number to remove as parameter. You can only add or remove one port at a time.

```
# ndd /dev/tcp tcp_extra_priv_ports
2049
4045
# ndd -set /dev/tcp tcp_extra_priv_ports_add 4444 5555
# ndd /dev/tcp tcp_extra_priv_ports
2049
4045
4444
# ndd -set /dev/tcp tcp_extra_priv_ports_del 4444
# ndd /dev/tcp tcp_extra_priv_ports
2049
4045
```

Analogous procedures apply to UDP extra privileged port.

6. Windows, buffers and watermarks

This section is about windows, buffers and watermarks. It is still work in progress. The explanations available to me were very confusing (sigh), though the Stevens [7] helped to clear up a few things. If you have corrections to this section, please let me know and contribute to an update of the page. Many readers will thank you!

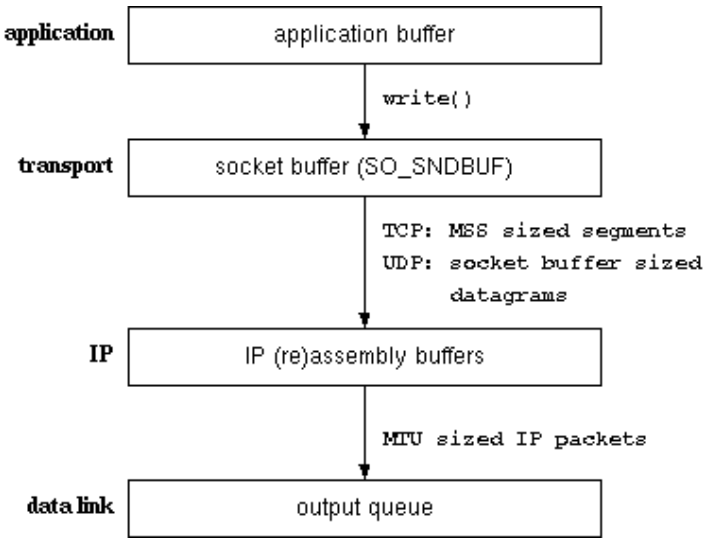

Figure 4: buffers and related issues

Here just a short trip through the network layer in order to explain what happens where. Your application is able to send almost any size of data to the transport layer. The transport layer is either UDP or TCP. The socket buffers are implemented on the transport layer. Depending on your choice of transport protocol, different actions are taken on this level.

TCP

All application data is copied into the socket buffer. If there is insufficient size, the application will be put to sleep. From the socket buffer, TCP will create segments. No chunk exceeds the MSS.

Only when the data was acknowledged from the peer instance, the data can be removed from the socket buffer! For slow connections or a slowly working peer, this implies a very long time some data uses up the buffer.

UDP

The socket buffer size of UDP is simply the maximum size of datagram UDP is able to transmit. Larger datagrams ought to elicit the `EMSGSIZE` error response from the socket layer. With UDP implementing an unreliable service, there is no need to keep the datagram in the socket buffer.

Please assume that there is not really a socket buffer for sending UDP. This really depends on the operating systems, but many systems copy the user data to some kernel storage area, whereas others try to eliminate all copy operations for the sake of performance.

Please note that for the reverse direction, that is receiving datagrams, UDP does indeed employ real buffering.

The IP layer needs to fragment chunks which are too large. Among the reasons TCP prechunks its segments is the need to avoid fragmentation. IP searches the routing tables for the appropriate interface in order to determine the fragment size and interface.

If the output queue of the datalink layer interface is full, the datagram will be discarded and an error will be returned to IP and back to the transport layer. If the transport protocol was TCP, TCP will try to resend the segment at a later time. UDP should return the `ENOBUFS` error, but some implementations don't.

To determine the MTU sizes, use the `ifconfig -a` command. The MTUs are needed for some calculation to be done later in this section. With IPv4 you can determine the MSS from the interface MTU by subtracting 20 Bytes for the TCP header and 20 Bytes for the IP header. Keep this in mind, as the calculation will be repeatedly necessary in the text following below.

```
$ ifconfig -a
lo0: flags=849<UP,LOOPBACK,RUNNING,MULTICAST> mtu 8232
        inet 127.0.0.1 netmask ff000000
hme0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
        inet 130.75.3.xxx netmask ffffff80 broadcast 130.75.3.255
ci0: flags=843<UP,BROADCAST,RUNNING,MULTICAST> mtu 9180
        inet 130.75.214.xxx netmask ffffff00 broadcast 130.75.214.255
        ether xx:xx:xx:xx:xx:xx
fa0: flags=842<BROADCAST,RUNNING,MULTICAST> mtu 9188
```

```
        inet 0.0.0.0 netmask 0
        ether xx:xx:xx:xx:xx:xx
el0: flags=843<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 130.75.215.xxx netmask ffffff00 broadcast 130.75.215.255
        ether xx:xx:xx:xx:xx:xx
```

I removed the uninteresting things. `hme0` is the regular 100 Mbps ethernet interface. The 10 Mbps ethernet interface is called `le0`. The `el0` interface is an ATM LAN emulation (lane) interface. `ci0` is the ATM classical IP (clip) interface. `fa0` is the interface that supports Fore's proprietary implementation of native ATM. Fore is the vendor of the installed ATM card. AFAIK you can use this interface to build PVCs or, if you are also using Fore switches, SVCs. You see an unconfigured interface there.

The buffer sizes for sending and receiving TCP segment and for UDP datagrams can be tuned with Solaris. With the help of the `netstat` command you can obtain an output similar but unlike the following one. The data was obtained on a server which runs a Squid with five dnsserver children. Since the interprocess communication is accomplished via localhost sockets, you see both, the client side and the server side of each dnsserver child socket.

```
$ netstat -f inet

 TCP
   Local Address        Remote Address    Swind Send-Q Rwind Recv-Q  State
-------------------- -------------------- ----- ------ ----- ------ -------
blau-clip.ssh        challenger-clip.1023 57344     19 63980      0 ESTABLISHED
localhost.38437      localhost.38436      57344      0 57344      0 ESTABLISHED
localhost.38436      localhost.38437      57344      0 57344      0 ESTABLISHED
localhost.38439      localhost.38438      57344      0 57344      0 ESTABLISHED
localhost.38438      localhost.38439      57344      0 57344      0 ESTABLISHED
localhost.38441      localhost.38440      57344      0 57344      0 ESTABLISHED
localhost.38440      localhost.38441      57344      0 57344      0 ESTABLISHED
localhost.38443      localhost.38442      57344      0 57344      0 ESTABLISHED
localhost.38442      localhost.38443      57344      0 57344      0 ESTABLISHED
localhost.38445      localhost.38444      57344      0 57344      0 ESTABLISHED
localhost.38444      localhost.38445      57344      0 57344      0 ESTABLISHED
```

The columns titled with `Swind` and `Rwind` contain values for the size of the respective send- and reception **windows**, based on the free space available in the receive **buffer** at each peer. The `Swind` column contains the **offered window** size as reported by the remote peer. The `Rwind` column displays the **advertised window** size being transmitted to the remote peer.

An application can change the size of the the socket layer **buffers** with calls to `setsockopt` with the parameter `SO_SNDBUF` or `SO_RCVBUF`. Windows and buffers are not interchangeable. Just remember: The buffers have a fixed size - unless you use `setsockopt` to change. Windows on the other hand depend on the free space available in the input buffer. The minimum and maximum requirements for buffer sizes are tunable **watermarks**.

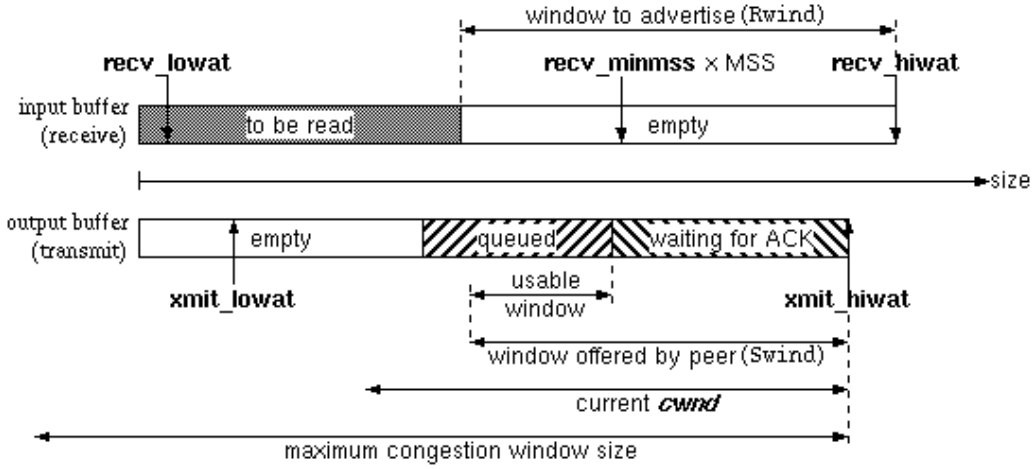

Figure 5: buffers, watermarks and window sizes.

Figure 5 shows the relation of the different buffers, windows and watermarks. I decided to let the send buffer grow from the maximum towards zero, which is just a way of showing things, and does probably not represent the real implementation. I left out the different socket options as the picture is confusing enough.

- **receive high watermark** `*_recv_hiwat`

  Effectively the size of the receive buffer. The receive high watermark tunes the transport layer socket buffer sizes on a kernel wide basis. The socket option `SO_RCVBUF` allows the dynamic change of the receive buffer size within the application on a per socket basis.

- **minimum number of maximum sized segments** `tcp_recv_hiwat_minmss` (TCP only)

The number of maximum sized segments which have to fit into a receive buffer. This is a k.o.-criterion for sizing buffers. It is not possible to set a receive buffer size which is unable to have at least this many segments fit into it.

- **receive low watermark** *not directly tunable* (TCP only)

  minimum amount of data in the input buffer to have a call to `select` or `poll` return the socket as readable. The socket option `SO_RCVLOWAT` allows the dynamic change of the receive low watermark on a per socket basis. With UDP, the socket is reported readable as soon as there is a complete datagram in the receive buffer.

- **transmit high watermark** `*_xmit_hiwat`

  Effectively the size of the send buffer. The send buffer high watermark tunes the transport layer socket buffer size on a kernel wide basis. The socket buffer can also be changed on a per-socket basis by using the `SO_SNDBUF` socket option within an application. Mind that for UDP the size of the output buffer represents the maximum datagram size.

- **transmit low watermark** `*_xmit_lowat`

  The amount of free space to be available in the send buffer to have `select` and `poll` report the socket writable. The socket option `SO_SNDLOWAT` allows a dynamic change of this size on a per-socket basis.

- **offered window** (TCP only)

  The offered window is the amount of free space available in the input buffer of the peer (how much we are allowed to send), relative to the acknowledged data. The window size is shown in the `Swind` column in the `netstat` output. From the offered window, the usable window is calculated, that is the amount of data which can be send as soon as possible. TCP never sends more than the minimum of the current congestion window and the offered window.

  ```
  to_send := MIN( cwnd, offered window )
  ```

- **advertised window** (TCP only)

  The advertised window is the free space available in the local input buffer advertised to the remote peer (how much we are willing to receive). The advertised window is shown in the `Rwind` column in the `netstat` output.

- ***cwnd*** (TCP only)

  The congestion window *cwnd* is used in TCP bulk data transfer with the *congestion avoidance* and *slow start* algorithms. The *cwnd* is initialized to 1 segment, and will grow with received acknowledgments. It will not exceed the maximum congestion window size. TCP will not send more than the minimum of the offered window and the current value of *cwnd*.

- **congestion window maximum size** `tcp_cwnd_max` (TCP only)

  Maximum size the congestion window may grow to. Usually, the maximum size is larger than the buffer size available, which is *not* a problem.

Squid users should note the following behavior seen with Solaris 2.6. The default socket buffer sizes which are detected during configuration phase are representative of the values for **tcp_recv_hiwat**, **udp_recv_hiwat**, **tcp_xmit_hiwat** and **udp_xmit_hiwat**. Also note that enabling the *hit object* feature still limits hit object size to 16384 byte, regardless of what your system is able to achieve.

> *Output from Squid 1.1.19 configuration script on a Solaris 2.6 host with the previously mentioned parameters all set to 64000. Please mind that these parameters do not constitute optimal sizes in most environments:*

```
checking Default UDP send buffer size... 64000
checking Default UDP receive buffer size... 64000
checking Default TCP send buffer size... 64000
checking Default TCP receive buffer size... 64000
```

Buffers and windows are very important if you link via satellite. Due to the daterate possible but the extreme high round-trip delays of a satellite link, you will need *very* large TCP windows and possibly the TCP timestamp option. Only RFC 1323 conformant systems will achieve these ends. In other words, get a Solaris 2.6. For 2.5 systems, RFC 1323 compliance can be purchased as a Sun Consulting Special.

Window sizes are important for maximum throughput calculations, too. As Stevens [4] shows, you cannot go faster than the window size offered by your peer, divided by the *round-trip time (RTT)*. The lower your RTT, the faster you can transmit. The larger your window, the faster you can transmit. If you intend to employ maximum window sizes, you might want to give **tcp_deferred_acks_max** another look.

The network research laboratory of the German research network did measurements on satellite links. The RTT for a 10 Mbps link (if I remember correctly) was about 500 ms. A regular system was able to transmit 600 kbps whereas a RFC 1323 conformant system was able to transmit about 7 Mbps. Only bulk data transfer will do that for you.

```
(1)   10 Mbps * 0.5 s = 5 Mbit = 625 KB
(2)   512 KB / 0.5 s = 1 MBps = 8 Mbps
```

```
(3)   64 KB / 0.5 s = 128 KBps = 1 Mbps
```

The *bandwidth-delay-product* can be used to estimate the initial value when tweaking buffer sizes. The buffers then represent the capacity of the link. If we apply the bandwidth-delay-product calculations to the satellite link above, we get the following results: Equation 1 estimates the buffer sizes necessary to fully fill the 10 Mbps link. Equation 2 assumes that the buffer sizes were set to 512 KB, which would yield 8 Mbps. Slight deviation in the experiment may have been caused by retransmissions. Finally, equation 3 estimates the maximum datarate we can use on the satellite link, if limited to 64 KB buffers, e.g. Solaris <= 2.5.1. The 1 Mbps constitute an upper limit, as can be seen by the measured 600 Kbps.

Application developers, especially those for web-based applications, should be aware of the implications of persistent connections. As long as HTTP/1.0 connection-per-transaction style is used by your application, depending on the size of the transaction data, you will not get any decent transmissions via satellite. For instance, the average web object is about 13 KByte in size, thus transmitting such an object on a connection-per-transaction basis will never get past TCP slow start. While this may or may not be a big deal with terrestrial links, but you will never be able to fill a satellite pipe to a satisfactorily degree. Doing things in parallel might help. Only when reaching TCP congestion avoidance you will see any filling of the pipe. You might also want to check out the unrelated **tcp_slow_start_initial** parameter.

A word of caution seems to be in order, when tuning the Solaris' TCP high watermarks: Starting with Solaris 2.6, setting **tcp_xmit_hiwat** or **tcp_recv_hiwat** near 65535 may have the side effect of turning on the wscale option, because these values are rounded *up* to multiples of MTU for each connection. In some cases you may not want to accidentally use wscale, because it may break something else in your setup such as IP-Filter. To avoid accidentally using wscale, you need to make sure that **tcp_xmit_hiwat** and **tcp_recv_hiwat** are both at least 1 MTU below 65535. For ethernet interfaces, 64000 is a good choice.

tcp_cwnd_max

default 32768, since 2.? 65535, recommended 65535 for Solaris <= 2.5.1
since 2.6: 262144 (finally!), no recommendations
Since 8: 1048576, no recommendations

This parameter describes the maximum size the *congestion window* can be opened. The *congestion window* is opened as large as possible with any Solaris up to 2.5.1. A change to this value is only necessary for older Solaris systems, which defaulted to 32768. The Solaris 2.6 default looks reasonable, but you might need to increase this further for satellite or long, fast links.

Though window sizes beyond 64k are possible, mind that the *window scale option* is only announced during connection creation and your maximum windows size is 1 GByte (1,073,725,440 Byte). Also, the *window scale option* is only employed during the connection, if *both sides* support it.

tcp_recv_hiwat

default 8192, recommended 16384 (see text), Cockroft 32768, maximum 65535
Solaris 2.6 *LFN bulk data transfer* 131071 or above (see text)
Since 8: 24576 (see text)

This parameter determines the maximum size of the initial TCP reception buffer. The specified value will be rounded up to the next multiple of the MSS. From the free space within the buffer the *advertised window size* is determined. That is, the size of the reception window advertised to the remote peer. Squid users will be interested in this value with regards to the socket buffer size the Squid auto configuration program finds.

The previous table shows an `Rwind` value of 63980 = 7 * 9140. 9140 is the MSS of the ATM classical IP interface (clip) in host `blau`. The interface itself uses a MTU of 9180. For the standard builtin 10 Mbps or 100 Mbps IPX ethernet, you get a MTU of 1500 on the outgoing interface, which yields an MSS of 1460. The value of 57344 in the next `Rwind` line points to the `lo0` (loopback) interface, MTU 8232, MSS 8192 and 57344 = 7 * 8192.

Starting with Solaris 2.6 values above 65535 are possible, see the *window scale* option from RFC 1323. Only if the peer host also implements RFC 1323, you will benefit from buffer sizes above 65535. If one host does not implement the *window scale option*, the window is still limited to 64K. The option is only activated, if buffer sizes above 64K are used.

For HTTP, I don't see the need to increase the buffer above 64k. Imagine servicing 1024 simultaneous connections. If both the TCP high watermarks of your system are tuned to 64k and your application uses the system's defaults, you would need 128M just for your TCP buffers!

Squid's configuration option `tcp_recv_bufsize` lets you select a TCP receive buffer size, but if set to 0 (default) the kernel value will be taken, which is configurable with the **tcp_recv_hiwat** parameter. A buffer size of 16K is large enough to cover over 70 % of all received webobjects on our caches.

Refer to **tcp_host_param** for a way to configure *special* defaults for a set of hosts and networks.

tcp_recv_hiwat_minmss

default 4, no recommendations

This parameter influences the minimum size of the input buffer. The reception buffer is at least as large as this value multiplied by the MSS. The real value is the maximum of **tcp_recv_hiwat** round up to the next MSS and **tcp_recv_hiwat_minmss** multiplied by the MSS, in other words, something akin to:

```
hiwat_tmp ~= ceil( tcp_recv_hiwat / MSS )
real_size := MAX( hiwat_tmp, tcp_recv_hiwat_minmss ) * MSS
```

That way, however bad you misconfigure the buffers, there is a guaranteed space for **tcp_recv_hiwat_minmss** full segments in your input buffer.

udp_recv_hiwat

default 8192, recommended 16384 (see text), maximum 65535

The highwater mark for the UDP reception buffer size. This value may be of interest for Squid proxies which use ICP extensively. Please read the explanations for **tcp_recv_hiwat.** Squid users will want at least 16384, especially if you are planning on using the (obsolete) *hit object* feature of Squid. A larger value lets your computer receive more seemingly simultaneous ICP PDUs.

If you see many dead parent detections in your `cache.log` file without cause, you might want to increase the receive buffer. In most environments an increase to 64000 will have a negligible effect on the memory consumption, as most application, including Squid, use only one or very few UDP sockets, and often in an iterative way.

Remember if you don't set your socket buffer explicitly with a call to `setsockopt()`, your default reception buffer will have about the mentioned size. Arriving Datagrams of a larger size might be truncated or completely rejected. Some systems don't even notify your receiving application.

tcp_xmit_hiwat

default 8192, recommended 16384 (see text), Cockroft 32768, maximum 65535
Solaris 2.6 *LFN bulk data transfer* 131071 or above (see text)
Since 8: 16384 (see text)

This parameter influence a heuristic which determines the size of the initial send window. The actual value will be rounded up to the next multiple of the MSS, e.g. 8760 = 6 * 1460. Also do read the section on **tcp_recv_hiwat**.

The table further to the top shows a `Swind` of 57344 = 7 * 8192. For the standard builtin 10 Mbps or 100 Mbps IPX ethernet, you get an MTU of 1500 on the outgoing interface, which yields a MSS of 1460.

Starting with Solaris 2.6 values above 65535 are possible, see the *window scale* option from RFC 1323. Only if the peer host also implements RFC 1323, you will benefit from buffer sizes above 65535. If one host does not implement the *window scale option*, the window is still limited to 64K.

I don't see the need to increase the buffer above 32K for HTTP applications. Imagine servicing 1024 simultaneous connections. If both TCP high watermarks of your system are tuned to 32K, you would need 64M just for your TCP buffers. Mind that the send buffer has to keep a copy of all unacknowledged segments. Therefore it is affordable to give it a greater size than the receive buffer. Again, 16K covers over 70 % of all transferred web objects on our caches, and 32K should cover 90 %.

Refer to **tcp_host_param** for a way to configure *special* defaults for a set of hosts and networks.

udp_xmit_hiwat

default 8192, recommended 16384, maximum 65535

This refers to the highwater mark for send buffers. May be of interest for proxies using ICP extensively. Please refer to the explanations for **tcp_xmit_hiwat.** Squid users will want 16384, especially if you are planning on using the *hit object* feature of Squid. Selecting a higher value for the transmission is not feasible.

Please remember that there exists no real send buffer for UDP on the socket layer. Thus, trying to send a larger amount of data than **udp_xmit_hiwat** will truncate the excess, unless the `SO_SNDBUF` socket option was used to extend the allowed size.

tcp_xmit_lowat

default 2048, no recommendations
Since 8: 4096, no recommendations

The current parameter refers to the amount of data which must be available in the TCP socket sendbuffer until `select` or `poll` return *writable* for the connected file descriptor.

Usually there is no need to tune this parameter. Applications can use the socket option `SO_SNDLOWAT` to change this parameter on a process local basis.

udp_xmit_lowat

default 1024, no recommendations

The current parameter refers to the amount of data which must be available until `select` or `poll` return *writable* for the connected file descriptor. Since UDP does not need to keep datagrams and thus needs no outgoing socket buffer, the socket will always be writable as long as the socket sendbuffer size value is greater than the low watermark. Thus it does not really make much sense to wait for a datagram socket to become writable unless you constantly adjust the sendbuffer size.

Usually there is no need to tune this parameter, especially not on a system-wide basis.

tcp_max_buf

default 262144, minimum 65536, no immediate recommendations
since 2.6 1048576, minimum 65536, no immediate recommendations

udp_max_buf

default 262144 (since 2.5), minimum 65536, no immediate recommendations

Finally found the explanations in the SUN TCP/IP Admin Guide. The current parameter refers to the maximum buffer size an application is allowed to specify with the `SO_SNDBUF` and `SO_RCVBUF` socket option calls. Attempts to use larger buffers will fail with a `EINVAL` return code from the socket option call. SUN recommends to use only the largest buffer necessary for any of your applications - that is, the supremum function, not the sum. Specifying a greater size does not seem to have much impact, if all your applications are well-behaving. If not, they may consume quite an amount of kernel memory, thus this parameter is also a kind of safety line.

A few odd remarks at this point, concerning the recommendations given for the transmission buffer sizes. I decreased the recommendations of Adrian Cockroft in favor of a more conservative memory consumption. Also, with an average HTTP object size of 13 KByte, you can expect to fit over 50 % of all objects into the transmission buffer. On the other hand, larger objects which are to be transmitted by a cache or webserver may suffer in certain circumstances. Furthermore, I should recommend a generic transmission buffer size which is double the reception buffer size. This recommendation bases on the fact that unacknowledged segments occupy the send buffer until they are acknowledged.

Here some more material from the SUN TCP/IP Admin Guide, kindly pointed out by Mr. Murphy. Refer to the SUN guide for a more detailed description of these parameters, and their respective applicability. Most noteworthy is **tcp_host_param**, which allows per host/network defaults regarding RFC 1323 TCP options.

tcp_wscale_always

Since 2.6: default 0

If the parameter is set (non-zero), then the *TCP window scale option* will always be negotiated during connection initiation. Otherwise, the scale option will only be used if the buffer size is above 64K. To take effect, both hosts have to support RFC 1323.

tcp_tstamp_always

Since 2.6: default 0

If the parameter is set (non-zero), then the *TCP timestamp option* will always be negotiated during connection initiation. The scale option will always be used if the remote system sent a timestamp option during connection initiation. To use the timestamp, both hosts have to support RFC 1323.

tcp_tstamp_if_wscale

Since 2.6: default 0

If the option is set (non-zero), the *TCP timestamp option* will be used in addition to the *TCP window scale option*, if the user has requested a buffer size above 64K, that is, if window scaling is active.

tcp_host_param_ipv6

Since 8: default is empty (this is a tabular value)

Refer to **tcp_host_param** for instructions on handling the table. The same rules apply except that the **ipv6** table is meant for IPv6, of course.

tcp_host_param

Since 2.6: default is empty (this is a tabular value)

This parameter represents a table which contains special TCP options to be used with a remote host or network. The table is configurable with the help of `ndd`, and empty by default. The following piece of code displays the contents of the table at various points, sets an entry and removes it again:

```
# ndd /dev/tcp tcp_host_param
Hash HSP      Address         Subnet Mask     Send       Receive    TStamp

# ndd -set /dev/tcp tcp_host_param '192.168.4.17 sendspace 262144 recvspace 262144'
# ndd /dev/tcp tcp_host_param
Hash HSP      Address         Subnet Mask     Send       Receive    TStamp
 125 62bae844 192.168.004.017 000.000.000.000 0000262144 0000262144      0
```

```
# ndd -set /dev/tcp tcp_host_param '192.168.4.17 delete'
# ndd /dev/tcp tcp_host_param
Hash HSP      Address         Subnet Mask     Send       Receive    TStamp
```

Use the *mask* command to supply a netmask for a network, and the *timestamp* command to supply the timestamp option. Fill this table from a startup script, if you want large default windows only for certain links (e.g. which go via satellite), but small windows for anything else. The content of this table takes precedence over the generic global values, if certain criteria are met:

- For the configured host or network, *sendspace* takes precedence over either **tcp_xmit_hiwat** or a user supplied value, iff the table value is larger.
- For the configured host or network, *recvspace* takes precedence over either **tcp_recv_hiwat** or a user supplied value, iff the table value is larger.
- A value of 1 for *timestamp* always tries to negotiate the TCP timestamp option for the configured host or network. However, a value of zero for *timestamp* may still negotiate the timestamp option, depending on the settings of **tcp_tstamp_always** and **tcp_tstamp_if_wscale**.

7. Tuning your system

This section evolved around tuning items, which were not directly related to the TCP/IP stack, but nevertheless play an important role in the tuning of any system. Refer to SUN's Solaris Tunable Reference Manual for more in-depth information.

7.1 Things to watch

Did you reserve enough `swap` space? You should have at least as much `swap` as you have main memory. If you have little main memory, even double your `swap`. Do not be fooled by the result of the `vmstat` command - read the manpage and realize that the small value for free memory shown there is (usually) correct.

With Solaris there seems to exist a difference between virtually generated processes and real processes. The latter is extremely dependent on the amount of virtual memory. To test the amount of both kinds of processes, try a small program of mine. Do start it at the console, without X and **not** as privileged user. The first value is the hard limit of processes, and the second value the amount of processes you can really create given your virtual memory configuration. Tweaking your `ulimit` values may or may not help.

7.2 General entries in the file `/etc/system`

The file `/etc/system` contains various very important resource configurable parameters for your system. You use these tunings to give a heavily loaded system more resources of a certain kind. Unfortunately a *reboot* is necessary after changing anything. Though one could schedule reboots after midnight, I advice against it. You should always check if your changes have the desired effect, and won't tear down the system.

Adrian Cockroft severely warns against transporting an `/etc/system` from one system onto another, even worse, onto another hardware platform:

Clean out your `/etc/system` when you upgrade.

The most frequent changes are limited to the number of file descriptors, because the socket API uses file descriptors for handling internet connectivity. You may want to look at the hard limit of filehandles available to you. Proxies like Squid have to count twice to thrice for each request: open request descriptors and an open file and/or (depending what squid you are using) an open forwarding request descriptors. Similar calculations are true for other caches.

You are able to influence the tuning with the reserved word `set`. Use a whitespace to separate the key from the keyword. Use an equals sign to separate the value from its key. There are a few examples in the comments of the file.

Please, before you start, **make a backup copy of your initial `/etc/system`.** The backup should be located on your root filesystem. Thus, if some parameters fail, you can always supply the alternative, original system file on the boot prompt. The following shows two typically entered parameters:

```
* these are the defaults of Solaris < 8
set rlim_fd_max=1024
set rlim_fd_cur=64
```

WARNING! SUN does not make any guarantees for the correct working of your system, if you use more file descriptors than 4096. Personally, my old fvwm window manager did quit working alltogether. In my case, I compiled it on a Solaris 2.3 or 2.4 system and transferred it always onwards to a 2.5 system. After re-compiling it on the new OS, it worked to my satisfaction.

If you experience SEGV core dumps from your `select(3c)` system call after increasing your file descriptors above 4096, you have to recompile the affected programs. Especially the `select(3c)` call is known to the Squid users for its bad tempers concerning the maximum number of file descriptors. SUN remarks to this topic:

> *The default value for FD_SETSIZE (currently 1024) is larger than the default limit on the number of open files. In order to accommodate programs that may use a larger number of open files with select(), it is possible to increase this size within a program by providing a larger definition of FD_SETSIZE before the inclusion of <sys/types.h>.*
>
> Note: This does not work as expected! See text below.

I did test this suggestion by SUN, and a friend of mine tried it with Squid Caches. The result was a complete success or disaster both times, depending on your point of view: If you can live with supplying naked women to your customers instead of bouncing logos of companies, go ahead and try it. If you really need to access file descriptors above 1024, **don't use `select()`, use `poll()` instead!** `poll()` is supposed to be faster with Solaris, anyway. A different source mentions that the redefinition workaround mentioned above works satisfactorily; not for me, my personal experiences warn against such an action.

At the pages of VJ are a some tricks which I incorporated into this paper, too. Personally I am of the opinion that the VJ pages are not as up to date as they could be.

Many parameters of interest can be determined using the `sysdef -i` command. Please keep in mind that many values are in **hexadecimal notation** without the 0x prefix. Another very good program to see your system's configuration is sysinfo, the program. Refer to the manpages how to invoke this program.

There is also the possibility to use a small helper script kindly supplied by Mr. Kroonma to have a look into some kernel variables with the help of the *absolute debugger* (adb). You can extend the script to suit your own needs, but you should know what you are doing. Refer to the manual page of the absolute debugger for details of displaying non-ulong datatype variables. If you don't know, what adb can do for you, hands off.

rlim_fd_cur

default 64, recommended 64 or 256
Since 8: default 256, no recommendations

This parameters defines the soft limit of open files you can have. The currently active soft limit can be determined from a shell with something like

```
ulimit -Sn
```

Use at your own risk values above 256, especially if you are running old binaries. A value of 4096 may look harmless enough, but may still break old binaries.

Another source mentions that using more than 8192 file descriptors is discouragable. It mentions that you ought to use more processes, if you need more than 4096 file descriptors. On the other hand, an ISP of my acquaintance is using 16384 descriptors to his satisfaction.

The predicate **rlim_fd_cur** <= **rlim_fd_max** must be fulfilled.

Please note that Squid only cares about the hard limit (next item). With respect to the standard IO library, you should not raise the soft limit above 256. Stdio can only use <= 256 FDs. You can either use AT&T'ssfio library, or use Solaris 64-bit mode applications which fix the stdio weakness. RPC prior to 2.6 may break, if more than 1024 FDs are available to it.

Also note that RPC prior to Solaris 2.6 may break, if more than 1024 FDs are available to it. Also, setting the soft limit to or above 1024 implies that your license server queries break (first hand experience - thanks Jens). Using 256 is really a strong recommendation.

rlim_fd_max

default 1024, recommended >=4096

This parameter defines the hard limit of open files you can have. For a Squid and most other servers, regardless of TCP or UDP, the number of open file descriptors per user process is among the **most important parameter.** The number of file descriptors is one limit on the number of connections you can have in parallel. You can find out the value of your hard limit on a shell with something like

```
ulimit -Hn
```

You should consider a value of at least 2 * **tcp_conn_req_max** and you should provide at least 2 * **rlim_fd_cur**. The predicate **rlim_fd_cur** <= **rlim_fd_max** must be fulfilled.

Use at your own risk values above 1024. SUN does not make any warranty for the workability of your system, if

you increase this above 1024. Squid users of busy proxies will have to increase this value, though. A good starting seems to be 16384 <= x <= 32768. Remember to change the Makefile for Squid to use `poll()` instead of `select()`. Also remember that each call of `configure` will change the Makefile back, if you didn't change `Makefile.in`.

Any decent application will incorporate code to increase its soft limit to a possibly higher hard limit. Please note (again) that Squid, as such an application, only cares about the hard limit.

maxphys

default 126976 (sun4m and sun4d), 131072 (sun4u), 57,344 (Intel),
1048576 (`sd` driver with wide-SCSI), 1048576 (SPARC storage array driver), no recommendations

A work-copy of this value is often stored in the mount structure or driver structure at the time it is attached. If a driver sees IO requests larger than this parameter, the requests will be broken down into appropriotely sized chunks. The file system may further fragment the chunks. A change might be conceivable, if your database server uses raw devices and issues large requests - mind that many of todays database usage paradigms result in many small chunked requests and will not speed up by increasing this value.

If working large chunked IO with UFS, you can additionally increase the number of cylinder groups and decrease the number of inodes per group (as there will be a few large files).

maxusers

default 249 ~= Megs RAM (Ultra-2/2 CPUs/256 MB), min 8, max 2048, no recommendations

This parameter determines the size of certain kernel data structures which are initialized at startup. Recent versions of Solaris derive most table sizes now from the amount of memory available, but there are still some dependent variables on this parameter, see **max_nprocs**, **maxuprc**, **ufs_ninode**, **ncsize** and **ndquot**. There is strong indication that the default for **maxusers** itself is being determined from the main memory in megs. It might also be a function of the available memory and/or architecture.

The greater you chose the number for **maxusers**, the greater the number of the mentioned resources. The relation in strictly proportional: A doubling of **maxusers** will (more or less) double the other resources.

Adrian Cockroft advises against a setting of **maxusers**. The kernel uses a lot of space while keeping track of the RAM usages within the system, therefore it might need to be reduced on systems with gigabytes of main memory. The point to change this parameter is whenever the automagically determined number of user processes is way too high, e.g. file servers, database servers, compute servers with few processes, or way too low.

pidmax

Since 8: default 30000, minimum 266, maximum 999999, no recommendations

Starting with Solaris 8, you can determine the number of the largest possible value for a `pid_t` the system can set. From this parameter, the kernel variable `maxpid` will be set *once* during startup. `maxpid` on the other hand cannot be set via `/etc/system`.

reserved_procs

Since 8: default 5, mininum 5, maximum MAXINT, no recommendations

This parameter is the mysterious difference between the number of all processes **max_nprocs** and the number of user processes **maxuprc**, and affects the number of system process table slots reserved for uid 0, e.g. `sched`, `pageout` and `fsflush`.

Though a change is not immanently recommended, increasing the number of root slots to 10 plus number of root processes might be considered, in order to provide root with a shell at times the system is uncapable of creating a user-level shell, e.g. run-away user-processes, fork-of-death, etc.

max_nprocs

default 10+**maxusers***16, minimum 266, maximum MIN(**maxpid**,65534), no recommendations

This is the systemwide number of processes available, user *and* system processes. You should leave sufficient space to the parameter **maxuprc**. The value of this parameter is influenced by the setting of **maxusers**.

The number is used to compute various further parameters (see below), including the DNLC cache, the quota structures, System-V semaphore limits, address translation table resources for sun4m, sun4d and Intel Solaris verions.

maxuprc

default **max_nprocs**-**reserved_procs**, minimum 1, maximum=default, no recommendations

This parameter describes the number of processes available to users. The actual value is determined from **max_nprocs** which is itself determined by **maxusers**. Adjustments to this parameter should be implemented by changing **max_nprocs** and/or **reserved_procs** instead.

npty

default 48, no recommendations

The parameter defines the maximum number of BSD ttys (`/dev/ptty??`) available. A few BSD networking things might need these devices. If you run into a limit, you may want to increase the number of available ttys, but usually the size is sufficient.

pt_cnt

default 48, min 48, max 3000, no recommendations
Since 8: remove from `/etc/system`

Solaris only allocated 48 SYSV pseudo tty devices (slave devices in `/dev/pts/*`). On a server with many remote login, or many open xterm windows you may reach this limit. It is of little interest to webservers or proxies, but of greater interest for personal workstations.

Starting with Solaris 8, the pseudo terminals are allocated dynamically, see docs.sun.com. Presetting the variable to some value disables the dynamic allocation.

vac_size

default 16384 (with **maxusers** 249), recommended: don't set

This parameter specifies the size of the virtual address cache. If a personal workstation with many open xterms and sufficient tty devices has a very degraded performance, this parameter might be too small. My recommendation is to let the system chose the correct value. The current value is determined by the size of **maxusers**.

ufs_ninode

default 4323 = 17***maxusers**+90 (with **maxusers** 249), min 226, max: see below,
Or 2.5.1: 4323 = **max_nprocs**+16+**maxusers**+64, (with **max_nprocs** 3994 and **maxusers** 249)
Since 2.6: 4323 = 4*(**max_nprocs**+**maxusers**)+320 (with **max_nprocs** 3994 and **maxusers** 249)
no immediate recommendations

ncsize

default 4323 = 17***maxusers**+90 (with **maxusers** 249), min 226, max: see below,
Or 2.5.1: 4323 = **max_nprocs**+16+**maxusers**+64, (with **max_nprocs** 3994 and **maxusers** 249)
Since 2.6: 4323 = 4*(**max_nprocs**+**maxusers**)+320 (with **max_nprocs** 3994 and **maxusers** 249)
no immediate recommendations

The first formula is taken from the *NFS Server Performance and Tuning Guide for SUN Hardware*, the second formula is taken from the *System Administration Guide, Volume II* and the third from an email on `squid-users`. I guess, in the end, after substituting all variables and interdependencies, they turn out more or less the same.

The **ufs_inode** parameter specifies the size of an inode table. The actual value will be determined by the value of **maxusers**. A memory-resident inode is used whenever an operation is performed on an entity in the file system (e.g. files, directories, FIFOs, devices, Unix sockets, etc.). The inode read from disk is cached in case it is needed again. **ufs_ninode** is the size that the Unix file system attempts to keep the list of idle inodes. As active inodes become idle, if the number of idle inodes increases above the limit of the cache, the memory is reclaimed by tossing out idle inodes.

The **ncsize** parameter specifies the size of the directory name lookup cache (DNLC). The DNLC caches recently accessed directory names and their associated vnodes. Since UFS directory entries are stored in a linear fashion on the disk, locating a file name requires searching the complete directory for each entry. Also, adding or creating a file needs to ensure the uniqueness of a name for the directory, also needing to search the complete directory. Therefore, entire directories are cached in memory. For instance, a large directory name lookup cache size significantly helps NFS servers that have a lot of clients. On other systems the default is adequate. The default value is determined by **maxusers**.

Every entry in the directory name lookup cache (DNLC) points to an entry in the inode cache, so both caches should be sized together. The inode cache should be at least as big as the DNLC cache. For best performance, it should be the same size in the Solaris 2.4 through Solaris 8 operating environments.

The upper bound for the inode cache is set by the amount of kernel memory used for inodes. The largest test value was 34906. Starting with Solaris 2.5.1, each inode uses 320 byte kernel memory. I was able to set my inode cache to 54688 on an 80 MB sun4m, and there are reports of an even larger settings of 128000 entries in the inode cache on a 1 GB machine.

The kernel will decrease the inode cache based on the main memory available, if too large, but it will **not** perform any magic for ridiculous large values. Your application could suffer from inode starvation, if the value is too large, and the inodes are not sufficiently recycled. You can check the current settings with the help of the `netstat -k inode_cache` command. The example shows a maximum size of 54688 entries:

```
$ netstat -k inode_cache
inode_cache:
```

```
size 947 maxsize 54688 hits 74 misses 1214 kmem allocs 947 kmem frees 0
```

Warning: Do not set **ufs_ninode** less than **ncsize**. The **ufs_ninode** parameter limits the number of inactive inodes, rather than the total number of active and inactive inodes. With the Solaris 2.5.1. to Solaris 8 software environments, **ufs_ninode** is automatically adjusted to be at least **ncsize**. Tune **ncsize** to get the hit rate up and let the system pick the default **ufs_ninode**.

I have heard from a few people who increase **ncsize** to 30000 when using the Squid webcache. Imagine, a Squid uses 16 toplevel directories and 256 second level directories. Thus you'd need over 4096 entries just for the directories. It looks as if webcaches and newsserver which store data in files generated from a hash need to increase this value for efficient access.

You can check the performance of your DNLC - its hit rate - with the help of the `vmstat -s` command. Please note that Solaris 7 re-implemented the algorithm, and thus doesn't have the `toolong` entry any more:

```
$ vmstat -s
... 1743348604 total name lookups (cache hits 95%) 32512 toolong
```

Up to Solaris 7, only names less than 30 characters are cached. Also, names too long to be cached are reported. A cache miss means that a disk I/O *may* be needed to read the directory (though it might still be in the kernel buffer cache) when traversing the path name components to get to a file. A hit rate of less than 90 percent requires attention. Since only short names are cached in Solaris version prior to 7, such a behavior would call for putting Squid cache disks or News spool disks onto partitions of their own (always a recommended feature for various reasons), and, more importantly, use a mount point in the root directory with a short name, e.g. `/disk1`. `/var/spool/cache` just might be short enough for Squid.

Solaris 7 re-implemented the DNLC algorithm. Now, memory is allocated dynamically, and path names with more than 30 characters are cached, too. Mr. Storm pointed to Adrian Cockroft answers a reader's question on Sun World Online Letters Section:

> *You can set the DNLC to be as big as you like. You should benchmark Solaris 7 as it has a new, faster DNLC implementation that has the extra feature of knowing that a directory is totally cached in the DNLC, so it doesn't need to scan the disk to ascertain that a new filename isn't already in use.*

Solaris 8 6/00, further enhances the DNLC, see the System Administration Supplement for enlightenment. The improved DNLC is now capable of caching negative hits, that is, to verify the non-existence of a file. I reckon that there be cache coherence protocols employed, so an application polling for the existence of a lock file will be notified as soon as possible.

dnlc_dir_enable

Since 8 6/00: default 1, recommended: don't touch

The switch enables the DNLC for large directories. There is no need to touch, but if problem occur, then set this variable to 0, to turn of the caching of large directories.

dnlc_dir_min_size

Since 8 6/00: default 40, minimum 0, maximum MAXUINT, recommended: don't touch

dnlc_dir_max_size

Since 8 6/00: default MAXUINT, minimum 0, maximum MAXUINT, recommended: don't touch

MAXUINT may have different concrete values, depending on the kernel running in 32 bit or 64 bit mode.

The **dnlc_dir_min_size** places a minimum limit on the directories which will eventually be cached. It looks as if the default value is a balance between the overhead of setting up the cache for the directory, and by-passing the cache. It is one of the usual problems that caching comes not for free. For this reason, it is strongly suggested not to decrease the default. If performance problems occur when caching small directories, increase the minimum default. From the System Administration Supplement:

> *Note that individual file systems might have their own range limits for caching directories. For instance, UFS limits directories to a minimum of **ufs_min_dir_cache** bytes (approximately 1024 entries), assuming 16 bytes per entry.*

If performance problems occur with large directories, then enforce a limit on the cachable directory using **dnlc_dir_max_size**. The **dnlc_dir_enable** parameter might be another switch to disable the new DNLC of (overly) large directories.

bufhwm

default 2 % of main memory, no immediate recommendations, maximum 20 %

Now, considering the SVR3 buffer cache described by Maurice Bach [11], this parameter specifies the maximum memory size allowed for the kernel buffer cache. The 0 value reported by `sysinfo` says to take 2 % of the main

memory for buffer caches. `sysdef -i` shows the size in bytes taken for the buffer cache.

Refer to the *NFS Server Performance and Tuning Guide for SUN HW* for further documentation on this parameter. I have seen Squid admins increasing this value up to 10 %, also a recommendation for dedicated NFS servers with a relatively small memory system. On a larger system, the **bufhwm** variable may need to be limited to prevent the system from running out of the operating system kernel virtual address space.

The buffer cache is used to cache inode, indirect block, and cylinder group related disk I/O only. If you change this value, you have to enter the number of kByte you want for the buffer cache. Please keep in mind that you are effectively 'double buffering', if you increase this value in conjunction with a proxy-cache like Squid.

If you have your system accounting up and running, you can check and monitor your buffer cache with the `sar -b` command - check with the manual page on how to run `sar`. The numbers in the columns titled as `%rcache` and `%wcache` are reported for the read hit rate and write hit rate respectively. You need to tune your system, if your read hit rate falls below 90 % and/or your write hit rate falls below 65 %.

physmem

default number of available pages excluding kernel core and data, recommendation: don't touch

This is more or less a debug or test value to simulate a system with less memory than actually available.

lwp_default_stacksize

default 8192, 16384 for sun4u in 64bit mode, recommendation: don't touch

This parameter affects the size of the stack of a kernel thread *(light weight process, LWP)* at the time of its creation. Increasing this value will result in almost every kernel thread to use a larger stack, most of the time eating memory resources without using them. If your system panic due to running out of stack space, chances are that there is something wrong with your application.

ndquot

default 6484, no recommendations

This parameter specifies the size of the quota table. Many standalone webservers or proxies don't use quotas.

nstrpush

default 9, no recommendations

This parameter determines how many STREAMS modules you are allowed to push into the Solaris kernel - I guess this is a per user or per process count. The only application of widespread use which may need such a kernel module is `xntp`. Even with other modules pushed, usually you have sufficient room and no need to tweak this parameter.

strmsgsz

default 65536, no recommendations

This parameter determines the maximum size of a message which is to be piped through the SYSV STREAMS.

strctlsz

default 1024, no recommendations

The maximum size of the control part of a STREAMS message.

autoup

default 30, no (immediate) recommendations

tune_t_fsflushr

default 5, no (immediate) recommendations

The **autoup** value determines the maximum age a modified memory page. The `fsflush` kernel daemon wakes up every five seconds as determined by the **tune_t_fsflushr** interval. At each wakeup, it checks a portion of the main memory - the quotient of **autoup** divided by **tune_t_fsflushr**. The pages are queued to the `pageout` kernel daemon, which forms it into clusters for faster write access. Furthermore, the `fsflush` daemon flushed modified entries from the inode caches to disk!

Some squid admins recommend lowering this value, because at high disk loads, the `fsflush` effectively kills the I/O subsystem with its updates, unless the stuff is flushed out fairly often. Steward Forster notes that this is justifiable, because squid writes disjoint data sets and rarely does multiple writes to the same disk block. If

```
/usr/proc/bin/ptime sync
```

reports the time spent for updating the disks above five seconds on several occasions, you can consider lowering **autoup** among several options. Please note that a larger **bufhwm** will take longer to flush. Also, the settings of **ufs_ninode** and **ncsize** have an impact on the time spent updating the disks. Setting the value too low has harmful impact on your performance, too.

There are also instances, where *increasing* the **autoup** makes sense. Whenever you are using synchronous writes like NFS or raw database partition, `fsflush` has little to do, and the overhead of frequent memory scans are a hindrance. Refer to Adrian Cockroft [2] for a more detailed enlightenment on the subject. I never claimed that tweaking your kernel is easy nor foolproof.

use_mxcc_prefetch

default 0 (sun4d) or 1 (sun4m), recommended: see text

Adrian Cockroft explains in What are the tunable kernel parameters for Solaris 2? this parameter. The parameter determines the external cache controller prefetches. You have to know your workload. Applications with extensive floating point arithmetic will benefit from prefetches, thus the parameter is turned on on personal workstations. On random access databases with little or no need for float point arithmetic the prefetch will likely get into the way, therefore it is turned off on server machines. It looks as if it should be turned off on dedicated squid servers.

noexec_user_stack_log

Since 2.6: default ?, recommended: don't touch

noexec_user_stack

Since 2.6: default 0, recommended: see CERT CA-98.06, or DE-CERT. Limited to sun4[mud] platforms!
Warning: This option might crash some of your application software, and endanger your system's stability!

By default, the Solaris 32 bit application stack memory areas are set with permissions to read, write and execute, as specified in the SPARC and Intel ABI. Though many hacks prefer to modify the program counter saved during a subroutine call, a program snippet in the stack area can be used to gain root access to a system.

If the variable is set to a non-zero value, the stack defaults to read and write, but not executable permissions. Most programs, but not all, will function correctly, if the default stack permissions exclude executable rights. Attempts to execute code on the stack will kill the process with a SIGSEGV signal and log a message in `kern:notice`. Program which rely on an executable stack must use the `mprotect(2)` function to explicitly mark executable memory areas.

Refer to the System Administration Guide for more information on this topic. Admins which don't want the report about executable stack can set the **noexec_user_stack_log** variable explicitly to 0.

Also note that the 64 bit V9 ABI defaults to stacks without execute permissions.

priority_paging

Since 7 or 2.6 with patch >= 105181-09 applied: default 0, recommended 1
Since 8: remove from `/etc/system`

Priority paging is an advanced memory paging technique which enhances the responsiveness of the system. If the file system is used heavily, Solaris may suffer from the file system cache stealing pages from applications. High performance clusters almost always benefit from the priority paging. The more memory you have, the better it is to actively avoid swapping.

Please refer to Priority Paging page by Richard McDougall, Triet Vo, and Tom Pothier. The paper rumours about an appropriate kernel patch for Solaris 2.5.1.

There is one drawback, though, or a feature for some of us: If you data has the executable bit set, it can fool the virtual memory management into believing it is treating a real executable, and thus will not engage priority paging for that data.

The Solaris 8 operating environment introduces a new file system caching architecture, which subsumes the Solaris 7 Priority Paging functionality. The system variable priority_paging should not be set in the Solaris 8 operating environment, and should be removed from the directory `/etc/system` when systems are upgraded to the Solaris 8 operating environment.

tcp:tcp_conn_hash_size

default 256, recommended: increase on busy servers

The tcp connection hash size determines the size of the table where Solaris keeps all interesting information like RTO, MSS, windows and states on any TCP connection. You can check the current content of the table with the `ndd` command:

```
$ ndd /dev/tcp tcp_conn_hash
tcp_conn_hash_size = 256
  TCP      dest           snxt     suna     swnd       rnxt     rack     rwnd
rto   mss   w sw rw t recent   [lport,fport] state
251 f5bcf2a8 130.075.003.xxx 204a5e77 204a5e77 0000032120 e6255721 e6255721 0000034752
02000 01448 1 00 00 1 002a16c0 [22, 1022] TCP_ESTABLISHED
```

The default size is printed when investigating the table. If you have a busy server, you might want to consider

increasing the table's size. Mr. Storm reports that SUN increases the hash size up to 262144 for web server benchmarks.

nfssrv:nfs_portmon

default 0, recommended: read text, NFS only

If the value is set to 1, the NFS service daemon by places the restriction on the client to use a privileged port, see `nfsd(1m)`. It is said to make it a little more difficult to abuse Leendert's NFS shell, if the server is thus set up.

Some services use a multitude of caches files like Squid or some News server where names (URLs or articles) are mapped by a hash function to a shallow directory tree, helping the buffer cache and inode caches of the host file system (compared to using unlimited subdirectories like the CERN cache). As well-known in software engineering, the speedup by using *the right algorithm* usually far exceeds anything you can achieve by fiddling with the hardware or tweaking system parameters. Still, the services can be helped by proper tuning of **ncsize** and **ufs_ninode**.

7.3 System V IPC related entries

Many applications still use the (old) SYSV IPCs. The System V IPC can be ordered into the three separate areas message queues, shared memory and semaphores. With Solaris you have an easier and faster API to achieve the same ends with *Unix sockets* or FIFOs, shared memory through *memory maps*, see mmap(2), and *file locks* instead of semaphores. Due to the reduced need for System V IPC, Solaris has decreased the resources for System V IPC drastically. This is o.k. for stand alone servers, but personal workstations may need increased resources.

In some cases large database applications or VRML viewer use System V IPC. Thus you should consider increasing a few resources. The active resource can be determined with the `sysdef -i` command. Relevant for your inspection are the parts rather at the end, all having IPC in their names.

At first glance, the System V IPC resources for message queues and semaphores seem to be disabled by default. This is not true, because the necessary modules are loaded dynamically into the kernel as soon as they are referenced. The default System V shared memory uses 1 MB main memory. Proxy and webserver may even want to decrease this value, but database servers may need up to 25 % of the main memory as System V shared memory.

```
* personal workstations using mpeg_play, or vic
set shmsys:shminfo_shmmax=16777216
```

The entries in `/etc/system` for all System V IPC related informations contains the prefix `msgsys:msginfo_` for message queues, the prefix `semsys:seminfo_` for semaphores, and the prefix `shmsys:shminfo_` for shared memory. After the prefixes starts the resource identifier, all lower case letters, for the corresponding value displayed by the `sysdef` command, e.g. `shmmax` for the value of `SHMMAX`. The meaning of the parameters can be obtained from any programming resource on System V ICP, e.g. Stevens' [3]. If anything, you only need to change the value for `SHMMAX`.

7.4 How to find further entries

There are thousands of further items you can adjust. Every module which has a device in the `/dev` directory and a module file somewhere in the kernel tree underneath `/kernel` can be configured with the help of `ndd`. Whether you have to have superuser privileges depends on the access mode of the device file.

There is a way to get your hands on the names of keys to tweak. For instance, the System V IPC modules don't have a related device file. This implies that you cannot tweak things with the help of `ndd`. Nevertheless, you can obtain all clear text strings from the module file in the kernel.

```
strings -a /kernel/sys/shmsys # possible
nm /kernel/sys/shmsys # recommended
```

There is a number of strings you are seeing. Most of the strings are either names of function within the module or clear text string passages defined within. Strings starting with `shminfo` are the names of user tunable parameters, though. Now, how do you separate tunable parameters from the other stuff? I really don't know. If you have some knowledge about Sun DDI, you may be able to help me to find a recommendable way, e.g. using `_info`(9E) and `mod_info`.

The interesting part, though, is to configure devices and modules with the SUN supported way to do things, and that means using `ndd`. Please refer to the `ndd` section on how to use `ndd` for changing values non-permanently. Remember, if you want to know what names there are to tweak, use the question mark special parameter.

Of course, you can only change entries marked for read and write. If you are satisfied with your settings, and want to store the configuration as a default at boot time, you can enter your preferred values into the `/etc/system` file. Just prefix the key with the module name and separate both with a colon. You did see this earlier the System V IPC page, and the same will be shown for 100 Mbit ethernet.

8. 100 Mbit ethernet and related entries

This section focuses on the `hme` fast ethernet interface, but some ways to do things may be applicable to other interface cards, too. Please refer to the SUN Platform Notes: The hme Fast Ethernet Driver for a detailed introduction of the handling of the fast ethernet device. Refer to that document for the use of **lance_mode**, **pace_size** and **ipg0** through **ipg2**.

8.1 The hme interface

The current section can only be regarded as a quick introduction into a more complex theme. The focus is on the selection of the best performing data mode when inter-operating with switches and employing auto-negotiation. Refer to the Which network cards support full duplex (SUN FAQ) for an overview which interfaces support what data mode. Users of a Solaris 2.5.1 have to use Patch for 2.5.1 and auto negotiation, in order to use auto negotiation successfully. The release notes of the patch state:

> *NOTE2: For devices that do not advertise auto-negotiation and advertise 10-full-duplex and 10-half-duplex, hme will first select the 10-half-duplex. However, one can force it to 10-full-duplex (if desired).*

In order to check the current setting of your 100 Mbit interfaces, you have to use `ndd`. If you system is a 2.5.1, and unpatched, only rely on the data the switch, hub or router is giving you. You should make a special issue of back-checking the values obtained from your Solaris system with whatever kind of link-partner you are connected to.

instance

default: 0, see text

If there is just one hme interface installed in your system, `ndd` will auto-magically select the correct one. If there is more than one 100 Mbit interface card installed in your system, you have to select the appropriate card you want to inspect or modify. First check the file `/etc/path_to_inst` in order to identify the interface. Use that instance number, and set the **instance** parameter of the hme driver. Now all further modification or inspections will apply to just that particular interface.

Please note the importance of the **instance** parameter, if you have more than one card of the same kind installed. All inspections or modifications to parameters described below are depending on the setting of **instance**. The following set of read-only parameters allows you to inspect the behavior of the interface:

link_status (read-only)

default: 0 or 1

With the help of the **link_status** parameter you can determine whether your link is up or down. A value of 0 means that the link is down, a value of 1 that the link is up.

link_speed (read-only)

default: 0 or 1

This parameter lets you determine the speed which has been selected for the interface. The content is only valid, if the link is up. A value of 0 implies 10 Mbps, a value of 1 means 100 Mbps.

link_mode (read-only)

default: 0 or 1

The **link_mode** shows the duplex mode the link employs. The content is only valid, if the link is up. A value of 0 means that half-duplex is used, a value of 1 implies full-duplex. If you are detecting half-duplex mode, and you are sure that this is unwanted, you will need to take some of the steps described below.

transceiver_inuse (read-only)

default: 0 or 1

A value of 0 translates to "internal transceiver" and a value of 1 to the "external transceiver".

Check the content of the `link_*` values carefully. If you got all 1 values there, everything is working at optimum performance for an hme interface, and you might want to skip to the next section. On the other hand, if either Solaris or your link partner is telling you about sub-optimal performance like 10 Mbps and/or half-duplex mode, and you are absolutely sure that both partners, the Solaris host and its link partner, are able to perform better, you might need to tweak your setup. It is a well-known problem that auto negotiation of the link setup may fail.

You might first want to look, if your hardware thinks it is capable of supporting the modes you intend to select. Also, you might want to check what your interface things the link partner supports. There is a set of six values repeating for several values to check and one set of data to set. The asterisk `*` has the meaning of a wild card (like from a shell):

- **auto negotiation** `*autoneg_cap`

  The ability to automagically negotiate with the link partner the link speed and mode.

- **100Base-T4 ethernet** `*100T4_cap`

  The T4 standard allows for 100 Mbps transmissions over a lower quality wiring, but needs four pairs of wire as compensation. T4 is not supported by the internal transceiver.

- **100 Mbps full-duplex** `*100fdx_cap`

  The ability of transmitting at a speed of 100 Mbps and full-duplex, sending and receiving simultaneously.

- **100 Mbps half-duplex** `*100hdx_cap`

  The ability to transmit with 100 Mbps, but only one direction at a time, either sending or receiving.

- **10 Mbps full-duplex** `*10fdx_cap`

  The ability of transmitting at a speed of 10 Mbps and full-duplex, sending and receiving simultaneously.

- **10 Mbps half-duplex** `*10hdx_cap`

  The ability to transmit with 10 Mbps, but only one direction at a time, either sending or receiving.

If you replace the asterisk with the prefix **`lp`**_ (including the underscore), you get a set of six read-only variables, which describe the notion your interface has about its link partner. That is, the abilities advertised by your link partner, as seen from Solaris. Check the **lp_autoneg_cap** value first, because if it is 0, all the other **lp_*** values have an undefined meaning.

If you replace the asterisk with no prefix (just remove it), you get another set of six read-only variables. These variables describe the local transceiver abilities of the hardware. Please do not be too alarmed, if the transceiver reports to be able to support only half-duplex mode. According to SUN, the internal transceiver can support all capabilities. Thus you might still be able to configure full-duplex mode with the hme interface.

Finally, if you replace the asterisk with the prefix **`adv`**_ (including the underscore), you get yet another set of six variables, this time writable ones, which describe the capabilities the interface is to advertise to its link partner. After changing any values in this set, you have to shut the interface down with the `ifconfig` command, and start it up again, or temporarily disconnect the link cable. If more than one speed capability to advertise is activated, the items are priorized, highest priority first:

1. `adv_100fdx_cap`
2. `adv_100T4_cap`
3. `adv_100hdx_cap`
4. `adv_10fdx_cap`
5. `adv_10hdx_cap`

Table 1 shows the default values for the un-prefixed and **`adv`**_ prefixed sets. The table does not show the values for the **`lp`**_ set, as those are determined from the link partner capabilities. Please note that Solaris 2.5.1 and below default to half-duplex operations. In order to use auto negotiation, you have to use the patch mentioned above.

ability	*=""	*="adv_"
***autoneg_cap**	1	1
***100T4_cap**	0	0
***100fdx_cap**	0 (Solari < 2.6) 1 (Solari >= 2.6)	0 (Solari < 2.6) 1 (Solari >= 2.6)
***100hdx_cap**	1	1
***10fdx_cap**	0 (Solari < 2.6) 1 (Solari >= 2.6)	0 (Solari < 2.6) ? (Solari >= 2.6)
***10hdx_cap**	1	1

Table 1: Default values for the internal transceiver perceived abilities and advertisable abilities.

If you are experiencing trouble with auto negotiation, you will have to set explicitly the values supported for your interface card. For instance, if your link partner is not capable of auto negotiation, the correct speed for the link, and half-duplex mode (!) will be selected. But there are three ways to force your own choice:

1. setting values with the help of `ndd`
2. setting values in `/etc/system`
3. setting values in `hme.conf`

Please note that setting options with `ndd` only works until the next reboot. Also, you have to disconnect the link cable temporarily for a few seconds to initiate auto negotiation of the newly set capabilities. You should use `ndd` to test out a working set of capabilities, which you can manifest later in **either but not both** of the files mentioned above.

use_int_xcvr

default 0

If the default is active, the external transceiver will be used, if connected to the link. Otherwise the internal transceiver will be used. If you want to override an external transceiver, you can set this option to 1, and force the use of the internal transceiver.

adv_autoneg_cap

default 1, recommended 1, if possible

If you experienced severe problems with auto negotiation, you might want to try setting this value to 0. By using the zero value, you can force your preferred mode onto the hardward, but if your link partner does not support the chosen mode/speed combination, you might end up with nothing at all.

Usually the link partners like switches do auto negotiation, as well. For instance, if you want to force the use of 100 Mbps full-duplex, it may be necessary to set this parameter to 0 **and** configure your link partner hardware manually to 100 FDX. Also, only set **one** of the following parameters to 1, and all the others to 0. This is a last resort which always used to work for me.

adv_100T4_cap

default 0, no recommendations

The 100Base-T4 mode is only supported by an external transceiver, and usually not relevant for most of the hosts I know of.

adv_100fdx_cap

default 0, recommended 1, if possible
Since 2.6: 1

adv_100hdx_cap

default 1, recommended 1, if possible

The fdx parameter switches the advertising of the full-duplex mode capability, the hdx parameter of the half-duplex mode. If you experienced problems forcing your preferred mode, you can try to set the full-duplex parameter the opposite of the half-duplex value.

adv_10fdx_cap

default 0, recommended 1, if possible
Since 2.6: ?

adv_10hdx_cap

default 1, recommended 1, if possible

The latter parameters concern 10 Mbps speed capabilities to be advertised to a link partner. You'd probably prefer your server to work at a degraded performance, if your link partner and you happened to disagree on auto-negotiation, rather than not being able to reach it at all.

A few conditions on incorrectly working 100 Mbit interfaces result in a downgrade to 10 Mbit ethernet and/or half-duplex mode. Thus check at all available ends, if you are really getting the data rate you are expecting. The first hints about a misconfigured interface can be obtained with the `netstat -ni` input errors. Of course, good information can only be obtained at the link partner, if it happens to be a switch or router.

You have to be super-user to be able to tweak the `hme` device. If you are able to see any value of the `hme` interface with `ndd` as mere mortal user, you are suffering from a severe security hole. In that case check the access rights and ownership of the tools, devices and module files.

After you have determined a working set of special configurations, you can make the selection permanent by writing them into the `/etc/system` file. If you have more than one hme interface installed, you have to select the instance first. Otherwise, all modifications are reflected on *all* interfaces, sometimes the preferred way to initialize things.

In order to insert the values into `/etc/system`, you will have to prefix the **adv_*** values with **hme:hme_**. A typical entry in the `/etc/system` of a patched 2.5.1 hosts sets all capability advertisements. If the auto negotiation with the link partner works out, 100 Mbps full-duplex will be selected:

```
set hme:hme_adv_100fdx_cap=1
set hme:hme_adv_100hdx_cap=1
set hme:hme_adv_10fdx_cap=1
set hme:hme_adv_10hdx_cap=1
set hme:hme_adv_autoneg_cap=1
```

On the other hand, a Solaris 2.5 host must force the 100 Mbps mode in full-duplex. Additionally, the link partner has to disable its auto negotiation capability, and you have to manually instruct it to use 100 Mbps in full-duplex mode:

```
set hme:hme_adv_100fdx_cap=1
set hme:hme_adv_100hdx_cap=0
set hme:hme_adv_10fdx_cap=0
set hme:hme_adv_10hdx_cap=0
set hme:hme_adv_autoneg_cap=0
```

A 2.6 host should work correctly at optimum performance with its defaults, but it does not hurt to set the parameter like the patched 2.5.1 host. If you have more than one hme interface installed, and you need to configure them differently, first you have to select the **instance** as described above. Then you configure the parameters for that interface. Afterwards you can select a different **instance** and modify its configuration differently.

The other method to set the selected options permanently is to create a `hme.conf` file in the `/kernel/drv` directory. The contents of that file are not trivial, none of the kernel device configuration files are! Refer to the SUN Platform Notes: The hme Fast Ethernet Driver for the step by step guide of how to set up `hme.conf`.

If the third way to permanently set your options with the help `ndd` and a startup script looks tempting to you, you might want to consider appending the first startup script mentioned below. But please keep in mind that you have to shutdown and restart any `ndd` configured interfaces in order to have the options take effect.

8.2 Other problems

8.2.1 Multicast problems

If you are using the multicast backbone (MBone) of the internet, your ethernet interface, e.g. `be0` is probably the primary choice of the multicast interface. The interface will speak to your router as detailed in RFC 1112. Prior to Solaris 2.6, the ethernet driver allowed only for simultaneous participation in 64 multicast groups.

In previous versions of Solaris, when your multicast support ran out of space for group participation, IP believed from the error condition that the interface doesn't support any multicast whatsoever. Hence, it switched to link-level broadcasts for all multicast traffic - which does not inter operate with other hosts still using regular multicast traffic. Upgrading to the `hme` interface and Solaris >= 2.6 is said to have solved this particular problem.

8.2.2 Number of virtual interfaces

The number of virtual interfaces supported by any interface is finite, of course. The tunable parameter is **ip_addrs_per_if**.

8.2.3 Distinct MAC addresses for multiple physical interfaces

If you have more than one ethernet interface installed into your Solaris box, you will notice that SUNs by default use the MAC address of the first interface for all interfaces. Actually, it will use the MAC address burnt into the EEPROM of the motherboard. I cannot think of good reasons to do this, except for certain high-availability environments, so, if you want each interface to use its own MAC address, type as super-user:

```
eeprom local-mac-address\?=true
```

9. Recommended patches

It is utterly necessary to patch you Solaris system, if you didn't already do so! Have a look at the DFN CERT patch mirror or the original source from SUN. There may be a mirror closer to you, e.g. EUNet and FUNET have their own mirrors, if I am informed correctly.

In order to increase your TCP performance, security of websites and fix several severe bugs, do patch! Whoever still runs a Solaris below 2.5 should upgrade to 2.6 at least. Each new version of Solaris incorporates more new TCP features than the previous one, and bug fixes, too.

Please remember to press the Shift button on your netscape navigator while selecting a link. If the patch is not loadable, probably a new release appeared in the meantime. To determine the latter case, have a look at the directories of DFN CERT or SUN . The README file on the DNF-CERT server is kept without a version number and thus always up to date.

ip and ifconfig patch

- 103630-15 for Solaris 2.5.1 (README)
- 103169-15 for Solaris 2.5 (README)

tcp patch (only with ip patches)

- 103582-24 for Solaris 2.5.1 (README)
- 103447-10 for Solaris 2.5 (README)

hme patch

104212-13 for Solaris 2.5.1 (README)

Any system administrator should know the contents of SUN's patch page. Besides previously mentioned patches for a good TCP/IP performance, you should always consider the security related patches. Also, SUN recommends a set of further patches to complete the support for large IP addresses. You should really include any DNS related patch.

The SUN supplied patches to fix multicast problems with 2.5.1 are *incompatible* with the TCP patch. Unfortunately, you have to decide between an unbroken multicast and a fixed TCP module. Yes, I am aware that multicast is only possible via UDP, nevertheless the multicast patch replaces the installed TCP module. If you have problems here, ask your SUN partner for a workaround - he will probably suggest upgrading to 2.6.

10. Literature

The current section features a set of related material containing books, request for comments (RFCs), software and a multitude of links.

10.1 Books

[1]

Adrian Cockroft; *Sun Performance and Tuning;* SUN Microsystems Inc.; 1995; ISBN 0-13-149642-5. Regrettably only up to Solaris 2.4, but most information is still valid for current Solaris systems.
The Heise Verlag offers a German translation.

[2]

[must read] Adrian Cockroft; *Sun Performance and Tuning;* 2nd edition; SUN Microsystems Inc.; 04'1998; ISBN 0-13-095249-4. The improved version on performance and tuning, covers quick tips and Solaris 2.6 as well as Java server technologies.

[3]

W. Richard Stevens; *Advanced Programming in the UNIX Environment*; Addison-Wesley Publishing Company; Reading, MA; 1992; ISBN 0-201-56317-7.
A German translation is available as: Programmieren in der UNIX-Umgebung; ISBN 3-9319-814-8, 1995.

[4]

[must read] W. Richard Stevens; TCP/IP Illustrated, *Volume 1 - The Protocols*; Addison-Wesley Publishing Company; Reading, MA; 1994; ISBN 0-201-63346-9.
A German translation is available.

[5]

W. Richard Stevens; TCP/IP Illustrated, *Volume 2 - The Implementation*; Addison-Wesley Publishing Company; Reading, MA; 1995; ISBN 0-201-63354-X.
A German translation is available.

[6]

W. Richard Stevens; TCP/IP Illustrated, *Volume 3 - T/TCP, HTTP, NNTP, Unix Domain Sockets*; Addison-Wesley Publishing Company; Reading, MA; 1994; ISBN 0-201-63495-3.
A German translation is available.

[7]

W. Richard Stevens; Unix Network Programming, *Network APIs: Sockets and XTI*; Prentice-Hall Inc.; Upper Saddle River, NJ; 1998; ISBN 0-13-081081-9.
A German translation is *not* yet available.

[7b]

W. Richard Stevens; Unix Network Programming, *Network APIs: Sockets and XTI*; Prentice-Hall Inc.; Upper Saddle River, NJ; 1998; ISBN 0-13-490012-X.
A German translation is *not* yet available.

[8]

Brian Wong; *Configuration and Capacity Planning for Solaris Servers;* SUN Microsystems Inc.; 199?; ISBN 0-13-349952-9.
A book showing host- and peripheral technoligies. Contains hints on tuning. Eases the detection of hardware errors, because it explains about the workings of the hardware (you are then able to determine, if a misbehavior is a bug or a feature).

[9]

Andrew S. Tanenbaum;*Computer Networks;* I still use the 2nd edition; Prentice Hall Inc., 1989, ISBN 0-13162959-X (2nd) and 1996, ISBN 0-13349945-6 (3rd).

A German translation of the 2nd edition is availabe: *Computer Netzwerke;* Wolfram's Fachverlag, 1990, ISBN 3-925328-79-3.

[10]

Andrew S. Tanenbaum; *Modern Operating Systems;* Prentice Hall Inc., 1992, ISBN 0-13588187-0.

[11]

Maurice Bach;*Design of the Unix Operating System*; Prentice Hall, 1986, ISBN 0-13201799-7.
A German translation is available.

10.2 Internet resources

- Have a look at the Solaris FAQ.
- There is an RFC mirror from MERIT near you, too.
- The IANA assignments and related documents constitute resources you should also know about.
- Especially to this theme: TCP Performance Monitoring und -Tuning.
- SunSolve is featuring a tuning and performance related set of articles. Get information first hand from SUN.
- SUN's Solaris Tunable Reference Manual.
- Sun on the Net article about The slow start and delayed ACK.
- Sun article on priority paging by Richard McDougall, Triet Vo, and Tom Pothier.
- If you have not looked at SUN's documentation server already, all documentation on Solaris and SUN is available online in seven different languages.
- Several articles by Adrian Cockroft on Solaris Performance and Tuning:
  - Monitoring of webservers.
  - The slides concerning Solaris 2.5 Performance Update.
  - An overview on the Sunworld columns including the contributions by Adrian Cockroft concerning performance (he is the guru!).
  - *How does Solaris 2.6 improve performance stats and Web performance?*.
  - *What are the tunable kernel parameters for Solaris 2?*
- From Sun's *TCP/IP and Data Communications Administration Guide:*
  - *Setting Up and Administering TCP/IP Networks*
  - *TCP Large Window Support*
- SUN Platform Notes: The hme Fast Ethernet Driver
- SunWorld articles by Jim Mauro on System-V-IPC:
  - *Shared memory uncovered.*
  - *Setting our sights on semaphores.*
  - *Demangling message queues.*
- The Papers published by the Pittsburgh Supercomputing Center Networking Group constitute a highly recommended reading, for instance (see previous link for a complete list):
  - J. Semke, J. Mahdavi, M. Mathis, "Automatic TCP Buffer Tuning", *Computer Communication Review*, a publication of ACM SIGCOMM, volume 28, number 4, October 1998 (abstract).
  - J. Mahdavi, "Enabling High Performance Data Transfers on Hosts: (Notes for Users and System Administrators)", Technical note, Revised: December 1997.
  - M. Mathis, J. Semke, J. Mahdavi, T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm",*Computer Communication Review*, volume 27, number3, July 1997. (abstract).
  - M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," Internet Request for Comments 2018 (rfc2018.txt) October 1996.
- VJ contains descriptions, but may be a trifle outdated. A good starting point for hints on tuning.

10.3 RFC, mentioned and otherwise

RFCs mentioned in the text:

- RFC 793 Transmission Control Protocol (TCP, STD 7).
- RFC 1122 Requirements for Internet hosts - communication layers (STD 3).

- RFC 1123 Requirements for Internet hosts - application and support (STD 3), updated by RFC 2181.
- RFC 1323 TCP Extensions for High Performance.
- RFC 1700 Assigned numbers (STD 2), outdated, use the IANA assignments instead!
- RFC 1918 Address Allocation for Private Internets.
- RFC 2001 TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms.
- RFC 2018 TCP Selective Acknowledgment Options.
- RFC 2181 Clarifications to the DNS Specification.

Unmentioned, but important Internet resources, for Web services.
Compare with Duane Wessel's required reading list for Squid developers, and
W3C's change history of HTTP, or the HTTP protocol homepage. Links which are considered essential for the topic are marked dark green.

- Web related RFCs:
  - RFC 1737 Functional Requirements for Uniform Resource Names (URN).
  - RFC 1738 Uniform Resource Locators (URL). Updated by RFC 1808 and RFC 2368.
  - RFC 1808 Relative Uniform Resource Locators. Updates RFC 1738, updated by RFC 2368.
  - RFC 1945 Hypertext Transfer Protocol - HTTP/1.0.
  - RFC 2017 Definition of the URL MIME External-Body Access-Type.
  - RFC 2068 Hypertext Transfer Protocol - HTTP/1.1. Obsoleted by RFC 2616.
  - RFC 2069 An Extension to HTTP: Digest Access Authentication. Obsoleted by RFC 2617.
  - RFC 2109 HTTP State Management Mechanism.
  - RFC 2145 Use and Interpretation of HTTP Version Numbers.
  - RFC 2169 A Trivial Convention for using HTTP in URN Resolution.
  - RFC 2186 Internet Cache Protocol (ICP), version 2.
  - RFC 2187 Application of Internet Cache Protocol (ICP), version 2.
  - RFC 2192 IMAP URL Scheme.
  - RFC 2224 NFS URL Scheme.
  - RFC 2227 Simple Hit-Metering and Usage-Limiting for HTTP.
  - RFC 2255 The LDAP URL Format. Obsoletes RFC 1959.
  - RFC 2295 Transparent Content Negotiation in HTTP.
  - RFC 2296 HTTP Remote Variant Selection Algorithm -- RVSA/1.0
  - RFC 2368 The mailto URL scheme. Updates RFC 1738 and RFC 1808.
  - RFC 2384 POP URL Scheme.
  - RFC 2396 Uniform Resource Identifiers (URI): Generic Syntax.
  - RFC 2397 The "data" URL scheme
  - RFC 2483 URI Resolution Services Necessary for URN Resolution.
  - RFC 2518 HTTP Extensions for Distributed Authoring -- WEBDAV.
  - RFC 2616 Hypertext Transfer Protocol -- HTTP/1.1. Obsoletes RFC 2068.
  - RFC 2617 HTTP Authentication: Basic and Digest Access Authentication. Obsoletes RFC 2069.
  - RFC 2717 Registration Procedures for URL Scheme Names (BCP 35).
  - RFC 2718 Guidelines for new URL Schemes.
- TCP related RFCs:
  - RFC 793 Transmission Control Protocol (TCP, STD 7).
  - RFC 879 TCP maximum segment size and related topics.
  - RFC 896 Congestion control in IP/TCP internetworks.
  - RFC 1001 Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and methods (STD 19).
  - RFC 1002 Protocol standard for a NetBIOS service on a TCP/UDP transport: Detailed specifications (STD 19).
  - RFC 1072 TCP extensions for long-delay paths. Obsoleted by RFC 1323 and RFC 2018.
  - RFC 1122 Requirements for Internet hosts - communication layers (STD 3).

- RFC 1123 Requirements for Internet hosts - application and support (STD 3), updated by RFC 2181.
- RFC 1144 Compressing TCP/IP headers for low-speed serial links.
- RFC 1185 TCP Extension for High-Speed Paths. Obsoleted by RFC 1323.
- RFC 1263 TCP Extensions Considered Harmful.
- RFC 1323 TCP Extensions for High Performance. Obsoletes RFC 1072 and RFC 1185.
- RFC 1337 TIME-WAIT Assassination Hazards in TCP.
- RFC 1379 Extending TCP for Transactions -- Concepts.
- RFC 1644 T/TCP -- TCP Extensions for Transactions Functional Specification.
- RFC 2001 TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. Obsoleted by RFC 2581.
- RFC 2018 TCP Selective Acknowledgement Options. Obsoletes RFC 1072.
- RFC 2140 TCP Control Block Interdependence.
- RFC 2398 Some Testing Tools for TCP Implementors (FYI 33). Obsoletes RFC 1739.
- RFC 2415 Simulation Studies of Increased Initial TCP Window Size.
- RFC 2488 Enhancing TCP Over Satellite Channels using Standard Mechanisms (BCP 28).
- RFC 2525 Known TCP Implementation Problems.
- RFC 2581 TCP Congestion Control. Obsoletes RFC 2001.
- RFC 2582 The NewReno Modification to TCP's Fast Recovery Algorithm.

- DNS related RFCs:
  - RFC 1034 Domain names - concepts and facilities (STD 13) obsoleted by RFC 1065, RFC 2308; updated by RFC 1101, RFC 1183, RFC 1348, RFC 1876, RFC 1982, RFC 2065, RFC 2181, RFC 2308, RFC 2535.
  - RFC 1035 Domain names - implementation and specification (STD 13) updated by RFC 1101, RFC 1183, RFC 1348 (1706), RFC 1876, RFC 1982, RFC 1995, RFC 1996, RFC 2065, RFC 2181, RFC 2136, RFC 2137, RFC 2308, RFC 2535.
  - RFC 1101 DNS encoding of network names and other types. Updates RFC 1034, RFC 1035.
  - RFC 1183 New DNS RR Definitions. Updates RFC 1034, RFC 1035.
  - RFC 1464 Using the Domain Name System To Store Arbitrary String Attributes.
  - RFC 1706 DNS NSAP Resource Records.
  - RFC 1712 DNS Encoding of Geographical Location.
  - RFC 1713 Tools for DNS debugging (FYI 27).
  - RFC 1794 DNS Support for Load Balancing.
  - RFC 1876 A Means for Expressing Location Information in the Domain Name System (Updates RFC 1034, 1035).
  - RFC 1886 DNS Extensions to support IP version 6.
  - RFC 1982 Serial Number Arithmetic.
  - RFC 1995 Incremental Zone Transfer in DNS (Updated RFC 1035).
  - RFC 1996 A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY) (Updates RFC 1035).
  - RFC 2052 A DNS RR for specifying the location of services (DNS SRV).
  - RFC 2136 Dynamic Updates in the Domain Name System (DNS UPDATE, Update RFC 1035).
  - RFC 2137 Secure Domain Name System Dynamic Update.
  - RFC 2163 Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM).
  - RFC 2168 Resolution of Uniform Resource Identifiers using the Domain Name System.
  - RFC 2181 Clarifications to the DNS Specification.
  - RFC 2182 Selection and Operation of Secondary DNS Servers (BCP 16)
  - RFC 2219 Use of DNS Aliases for Network Services (BCP 17)
  - RFC 2230 Key Exchange Delegation Record for the DNS.
  - RFC 2308 Negative Caching of DNS Queries (DNS NCACHE). Obsoletes RFC 1034, Updates RFC 1034 and RFC 1035.
  - RFC 2352 A Convention For Using Legal Names as Domain Names.

    - RFC 2535 Domain Name System Security Extensions.
    - RFC 2536 DSA KEYs and SIGs in the Domain Name System (DNS).
    - RFC 2537 RSA/MD5 KEYs and SIGs in the Domain Name System (DNS).
    - RFC 2538 Storing Certificates in the Domain Name System (DNS).
    - RFC 2539 Storage of Diffie-Hellman Keys in the Domain Name System (DNS).
    - RFC 2540 Detached Domain Name System (DNS) Information.
    - RFC 2541 DNS Security Operational Considerations.
    - RFC 2606 Reserved Top Level DNS Names (BCP ??)
    - RFC 2671 Extension Mechanisms for DNS (EDNS0).
    - RFC 2672 Non-Terminal DNS Name Redirection.
    - RFC 2673 Binary Labels in the Domain Name System.
    - RFC 2694 DNS extensions to Network Address Translators (DNS_ALG).
- Further services:
    - RFC 768 User Datagram Protocol (UDP, STD 6)
    - RFC 792 Internet Control Message Protocol (ICMP, STD 5)
    - RFC 1112 Host extensions for IP multicasting (IGMP, STD 5)
    - RFC 2151 A Primer On Internet and TCP/IP Tools and Utilities (FYI 30).
      See also the Internet Measurement Tool Survey by CAIDA.

Also of interest in the regard of webservices may be a bunch of related drafts, partially expired, still sprouting with ideas. Compare with the IETF - Hypertext Transfer Protocol (HTTP) Working Group published documents and W3C's change history of HTTP:

- Access-restricted, HTTP/1.1 Cache Control Extension (draft-melve-cachecontrol-00.txt).
- HTTP Connection Management (draft-ietf-http-connection-00.txt).

More recently, the work of the IETF WREC (web replication and caching) working group was created. Its first effort deals with a taxonomy for terms related to replication services and cache services, including proxy services.

10.4 Further material

Software by SUN - no support offered!

- Have a look at the excellent GNU- and Freeware collection for Solaris systems. There is a mirror near you, too! The collections includes a proctool performance monitor package.
- In December 1998, Sun released SyMON 2.0, a complete rewrite and update of its system monitoring product, which includes a version of the rule set that Adrian developed as its health monitor. The release of SymbEL 3.* updates the SE toolkit.

11. Solaris' Future

This section will deal with forthcoming releases of the Solaris operating systems.

11.1 Solaris 7

Solaris 7 was announced the end of October 1998. There are many exciting new things in Solaris 7, and I am not talking about the 64 bit capabilities. The readers of this page are probably as interested, what bugs are fixed, extensions are implemented and features are available. Note, even though it is called *Solaris 7*, the `uname` command will still return 5.7, thus being staying compatible with many scripts which are in circulation.

- Inside Solaris 7 - SunWorld - November 1998
- Solaris 7 arrives - SunWorld - November 1998
- What's new in Solaris 7? - SunWorld - November 1998

The interested reader will, of course, want to skip through the pages mentioned above. For the impatient, here are the most interesting features for speeding up your webrelated services:

- **SACK**

  One feature you definitely want to read more about are the *selective acknowledgments (SACK)*, in accordance with

RFC 2018 - TCP Selective Acknowledgment Options. The availability of the SACK feature is negotiated during connection initiation. But the actual selective acknowledgment itself is sent from the data sink to the data source specifying the ranges of that data which was received correctly. The SACK option will lead to less data being retransmitted, as only lost or damaged segments need to be repeated.

- **mount option `noatime`**

  The new mount option allows to mount file systems without having to update the inodes at each access to any file. This will significantly help to speed up services like web caches or news servers, which do a lot of file IO with small files. Other than the creation and modification times, the access time is neither for web content nor news files necessary. You'd only want to mount your cache or news partitions that way, thus saving inode updates andl eaving more important inodes in the buffer cache memory.

- **priority paging**

  Priority paging is an advanced memory paging technique which enhances the responsiveness of the system, see the appropriate system tuning section. Also refer to Priority Paging page by Richard McDougall, Triet Vo, and Tom Pothier.

- **mount option `logging`**

  The UFS logging feature keeps a transaction log within the thus mounted partition. The advantage is an almost instantaneous filesystem check - which may take a considerable while with larger harddisks, e.g. 18 GB. The disadvantage is the additional time spent writing the transaction log. Due to the log being kept on the same physical disk, I wouldn't recommend it for cache nor news partitions.

- **stdio supports FDs up to 64k**

  The stdio library now understands about file descriptors above and beyond 256. Increasing the soft limit of concurrently open file descriptors will now be less contraproductive.

  You can try a little test program to check your FD limits. Remember to use the appropriate parameters for 64bit mode `-xtarget=ultra -xarch=v9`.

- **`poll()` and `select()` improvements**

  The `poll()` system call was improved to allow for even more file descriptors to be tested more often. The `select()` code in 64bit mode allows for 64k file descriptors.

11.2 Solaris 8

Solaris 8 did a lot of work in the TCP tunable section, aimed at further performance improvements for Internet servers and services.

Also, Solaris 8 offers IPv6 without needing to obtain extra packages. If enabled, IPv6 support is integrated into many regular places. Still, for a secure Solaris installation, it is recommended to neither install Kerberos nor IPv6.

If you are installing Solaris 8 for Intel, and you would like to use the stand-alone installation, boot from the 2nd CD-ROM. My installation with the web-installer failed frequently on different machines, and the stand-alone installation (my favourite, anyway) was the only way to get going.

Solaris 8 media kit now comes with many highly usable open source programs like gcc 2.95.2 (the SPARC optimizer might still be brain-dead, if using `-mcpu=ultrasparc`), perl 5.005_03, ghostscript 5.10, olvwm, rxvt, tiff, XPM, flex, bison, automake and many more.

If you upgrade to Solaris 8, clean out your `/etc/system`. Several of the parameters changed meaning, and should no longer be set in `/etc/system`. Look at **pt_cnt** and **priority_paging**.

I am still collecting more material about Solaris 8!

11.3 Solaris 9

TBD

12. Uncovered material

There are a bunch of parameters which I didn't cover in the sections above, but some of which may be worth looking at, among these **tcp_ip_abort_linterval**, **tcp_ip_notify_cinterval**, **tcp_ip_notify_interval**, **tcp_rexmit_interval_extra**, **tcp_sth_rcv_lowat**.

13. Scripts

For the important tweakable parameters exist startup scripts for Solaris. Only the first script is really necessary.

1. The first script changed all parameters deemed necessary and described in the previous sections. The file should be called something like `/etc/init.d/nettune` and you must link (hardlinks preferred, symbolic links are o.k.) `/etc/rcS.d/S31nettune` to the `init.d` file.

   > SUN recommends to run control scripts like `nettune` between `S69inet` and `S72inetsvr`.

   Please read the script carefully before installing. It is a rather straight-forward shell script. The piping and awking isn't as bad as it looks:

   - The script is also part of the YaSSP setup, and thus contains a few variables relating to YaSSP. Ignore them, if confused.
   - The initial check is sanity: Only Solaris systems should run this script.
   - There is no **stop** action for the script. Allowed actions include **start** to set parameters, and no argument, which just lists the current value of any parameter that the script would touch.
   - If there is a YaSSP configuration file, read it.
   - Always go for high security settings.
   - The first line sets the `PATH` to standard values and prints a message. For all messages which are not to contain a linefeed, we have to include a backslash c.
   - The `modlist` function changes a parameter to one or a list of values. Depending on the state of certain variables, only the status is display, the status is displayed before changed, or a value is changed silently.
   - The variable `$osver` is set with the operating system major and minor version number times ten: Solaris 2.6 AKA SunOS 5.6 will set `$osver` to 560 and Solaris 2.5.1 AKA SunsOS 5.5.1 will be counted as 551.
   - `$patch` looks into the installed kernel TCP module, because it mustn't be assumed that `/var` is already mounted. The result is either 0 for an unpatched system (or some error in the pipeline), or the applied TCP patch level. For non-2.5.1 systems, you have to change this line to your needs. All 2.5.1 system (Sparc, x86 and PPC) will be recognized.
   - The next `if` tree just prints a message about the patch found.
   - Depending on your OS version and the installed patch, we either have a single value of **tcp_conn_req_max** to tweak, or the twin values of **tcp_conn_req_max_q** and **tcp_conn_req_max_q0**.
   - The section about retransmission related tweaks follows.
   - Depending on your OS version and the installed patch level, we may have a parameter **tcp_slow_start_initial** to tweak. This parameter is said to have a mighty effect, if you have many BSD or MS clients complaining about a slow service.
   - The final sections are indicated by their respective comments and messages. There are a few things commented out, as they may not represent universally popular decisions. Please read the accompanying comment carefully, and decide for yourself, if you want to uncomment any of the options. Uncommenting should not hurt on leaf hosts, or hosts behind firewalls. Hosts acting as routers should reconsider.
   - The file ends in the change log.

   Always tune the parameters to *your needs*, not mine. Thus, examine the values closely.

2. The second script just changes the MTU of `le0` from the IPX to the IEEE 802.3 size. The meaning is shown further up. The script is not strictly necessary, and reports about odd behavior may have ceeded with a patched 2.5.1 or a 2.6.

   > Since I observed the erratic behavior only in a Solaris 2.5, I believe it has been fixed with patch 103169-10, or above. The error description reads "1226653 IP can send packets larger than MTU size to the driver."

   If you intend to go ahead with this script, the file is called `/etc/init.d/nettune2` and you need to create a link to it (hard or soft, as above) as `/etc/rc2.d/S90nettune2`. Please mind that GNU awk is used in the script, normal awk does not seem to work satisfactorily.

3. As this is the scripts section, I should remention the nifty script kindly supplied by Mr. Kroonmaa. It allow the user to check on all existing values for a network component (tcp, udp, ip, icmp, etc.). Previously, I did something similar in Perl, but nothing as sophisticated until I saw Mr. Kroonmaa's script. He is really talented with scripts.

4. There is a little helper, which lets you inspect a bunch of kernel related parameters. The script was supplied by Mr. Kroonmaa, and minimally modified by myself. It displays the contents of **physmem**, **minfree**, **desfree**, **lotsfree**, **fastscan**, **slowscan**, **maxpgio**, **tune_t_gpgslo**, **tune_t_fsflushr**, **autoup**, **ncsize**, **ufs_ninode**, **maxusers**, **max_nprocs**, **maxuprc**, **ndquot**, **nbuf**, **bufhwm**, **rlim_fd_cur**, **rlim_fd_max**, **nrnode**, **coredefault**, and is extensible to your needs. For the settings of these parameters, refer to Adrian Cockroft [2]. You should remember to

set the access rights to the script to only allow root and admins to use it, i.e. change it to group `adm`, and use mode 0750.

Another word of warning to this script, if you intend to use the *absolute debugger (adb)*, you'd better know what you are doing.

14. List of things to do

This section is not about things *you* have to do, but rather about items *which I* think of being in need to be reworked. Thus it is more a kind of meta-section.

- This page is definitely getting to large, even for me, who works on a LAN. I don't know about you out there, but I guess putting the sections into pages of their own might speed things up. On the other hand, if you say "print" on this page, you will get (almost) everything. What do you think?
- I am looking only at the downstream of web documents, which mostly will be TCP bulk data transfer. I haven't looked into the upstream direction (yet), as requests constitute interactive traffic, because they are not segment sized. Therefore, depending on further factors, requests might be hindered by delayed ACKs and NAGLE.
- The retransmission section needs to be reworked due to the erratic behavior of 2.5.1 TCP, if it loses the initial `SYN` segment. Solaris 2.6 is well-behaved in this regard. Also I should finish the few examples which show what is going on.
- Andres Kroonmaa asked for many `/etc/system` values to be put up that are in the meantime documented in SUN's Solaris Tunable Reference Manual.
- Mr. Kroonmaa suggested to extend on the on-the-fly kernel changes with the help of `adb`, especially those parameters, which are not accessible with `ndd`. Anybody out there more familiar with `adb`?
- Test what happens, if Solaris runs out of buffer memory for TCP or UDP. When does the ceiling apply, and does it start to swap out processes?
- POSIX.1g and TCP_KEEPALIVE implementation with Solaris 2.6 documentation.

[Solaris tuning] [TCP transactions] [SYS-V-IPC] [TCP rexmit] [Slow start] [Index] [JSV homepage]

Please send your suggestions, bugfixes, comments, and ideas for new items to solaris@sean.de
In hope of supplying useful information, Jens-S. Vöckler

Last Modified: Tuesday, 10-Apr-2001 16:11:22 MEST