

An Open Source iSCSI Initiator

Ken Davis

Sajid Zia

Solaris

Sun Microsystems



Copyright © 2007 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

Sun, Sun Microsystems, the Sun logo, Solaris, Open Solaris, Solaris Ready and the Solaris logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc.

Session Outline

- Introduction
- Architecture
- Integration
- Road Map
- Solaris Ready
- Open Solaris
- Code Overview
- Debugging
- Questions

Introduction

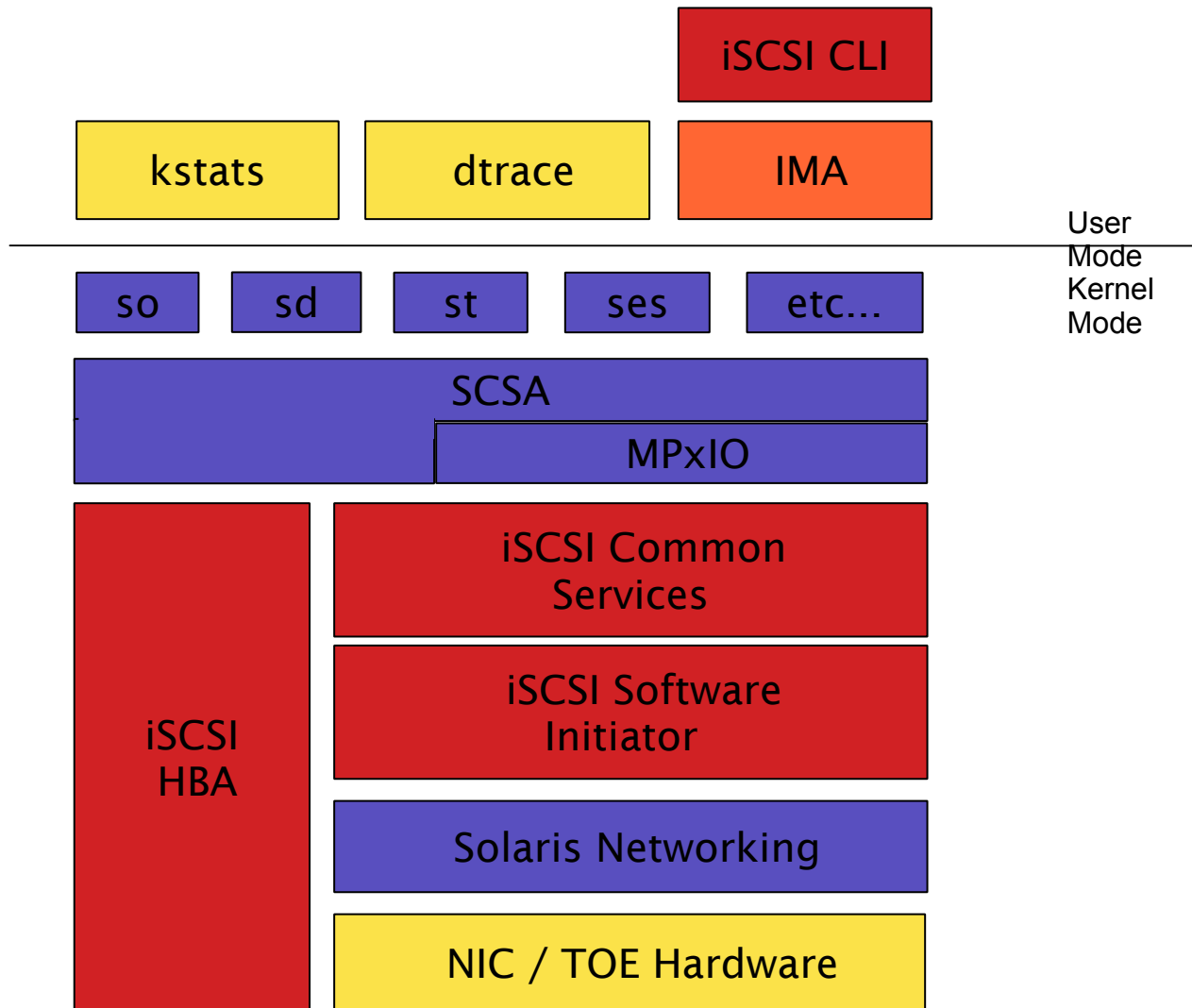
- Solaris iSCSI Software Initiator
 - Releases
 - Solaris Express
 - Solaris 10 Update 1
 - Platform Support
 - 64 bit – SPARC
 - 64 bit – AMD
 - 32 bit – i386
 - Any NIC / TOE Supported by Solaris

Introduction

- **Standard iSCSI Features**
 - **Discovery (static/ sendtargets / iSNS)**
 - **Login / Logout**
 - **Header/Data Digests**
 - **CHAP Authentication**
 - **Radius Support**
 - **Ipv6**
 - **CLI Management / Monitoring**
 - **2+ Terabyte (16 byte CDB support)**
 - **MpxIO Multipathing**
 - **Multiple Sessions per Target (MS/T)**



iSCSI Initiator Architecture



Kernel Mode Components

- **iSCSI Common Services**
 - **Discovery**
 - **Persistent Database**
 - **Discovery, Login, Authentication Parameters**
 - **Configuration Interface**
- **ISCSI Software Initiator**
 - **Login / Logout**
 - **Session and Connection Management**
 - **Persistent Device Naming**
- **Kernel Visibility / Management**
 - **ISCSI Management API – IMA**
 - **Kernel Statistics – kstat(1M)**
 - **Dynamic Tracing – dtrace (1M)**
 - **Modular Debugger – mdb (1)**

User Mode Components

- CLI Management Interface
 - Discovery Management
 - static / sendtargets / iSNS
 - Login Authentication Parameters
 - Monitoring
 - ISCSI Information (Login Parameters, etc)
 - SCSI Information (LUN info / OS device name)
 - IP Information (IP / Port, etc)
 - Utilizes IMA interface
- No Configuration file modifications needed
- No reboots needed



Solaris Integration

- devfs (7FS)
 - No reboot device addition / removal
- Volume Manager Support
 - SVM, ZFS, ...
- Clustering
- Network Features
 - IPMP- Fail Over / Load Spreading
 - IEEE 802.3ad
 - 10 Gbit NIC/TOE support

Multi-pathing

- SCSI Layer – MPxIO
 - Multiple Sessions per Target
 - Multiple Target Portal Groups
 - Initiator Portal Session Management
 - T-10 TPGS or /kernel/drv/scsi_vhci.conf binding
- Network Layer
 - IPMP Fail-Over / Load Spreading
 - IEEE 802.3ad
- Solaris Multi-Pathing blueprint

Open Solaris

- Code available at <http://www.opensolaris.org>
 - Released Q4CY05
 - iSCSI Community Forum

“ otevrěný 열린 مفتوح ανοικτό মুক্ত libre
 মুক্ত öppen open గణన 开放的
 開 オープン মুক্ত libero nyílt
 放的 வெளிப்படை açık : : : : livre offen
 的 OTKРЫTый

iSCSI Resources

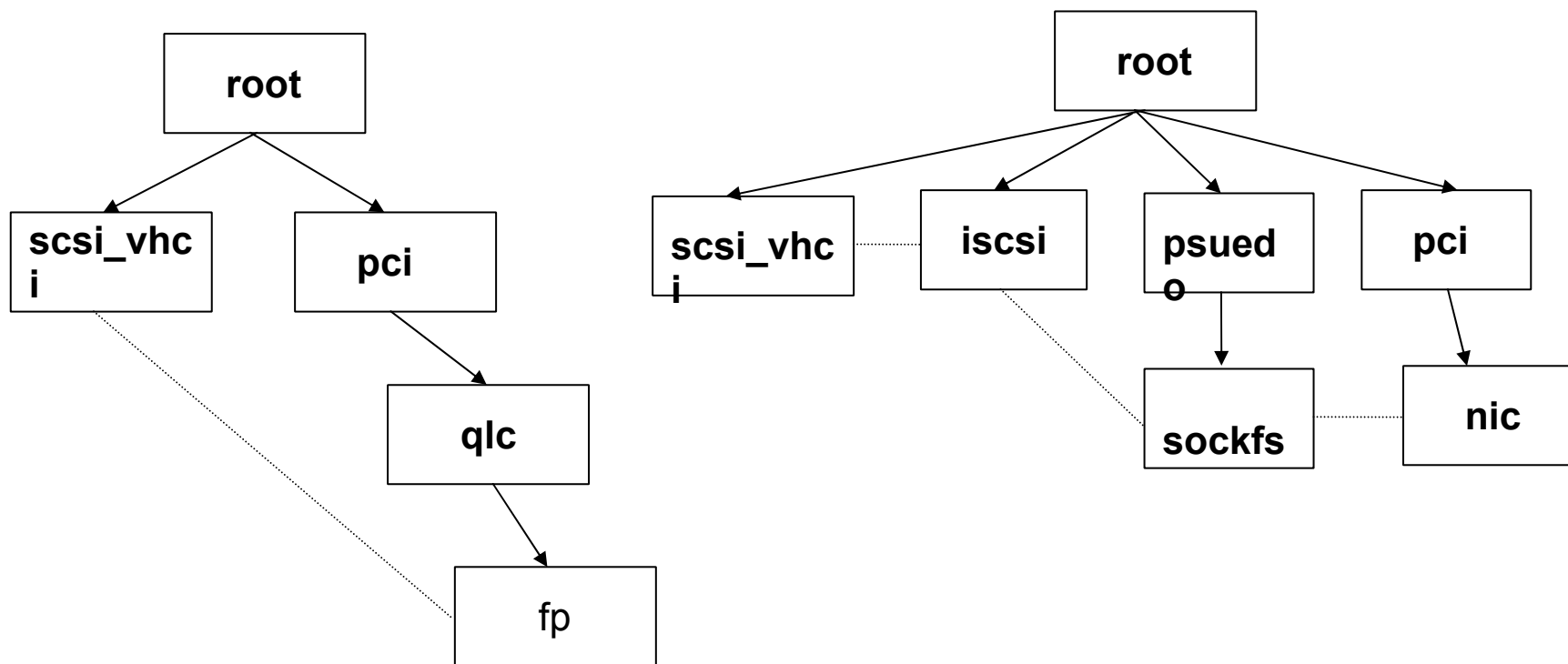
- iSCSI Initiator Documentation
 - <http://docs.sun.com>
- Solaris Ready
 - <http://www.sun.com/solarisready>
- Solaris Express
 - <http://www.sun.com/solarisexpress>
- Open Solaris
 - <http://www.opensolaris.org>

Solaris Initiator Design

- File Layout
- IO Model
- State Flows
- Connection Flows
- Cmd Flows
- IO Code Flow
- Discovery
- Enumeration
- Debugging

Where iSCSI lives in /devices?

- We can not be a pseudo driver, have to be off root due to NDI code



File Layout

iscsi_if.h

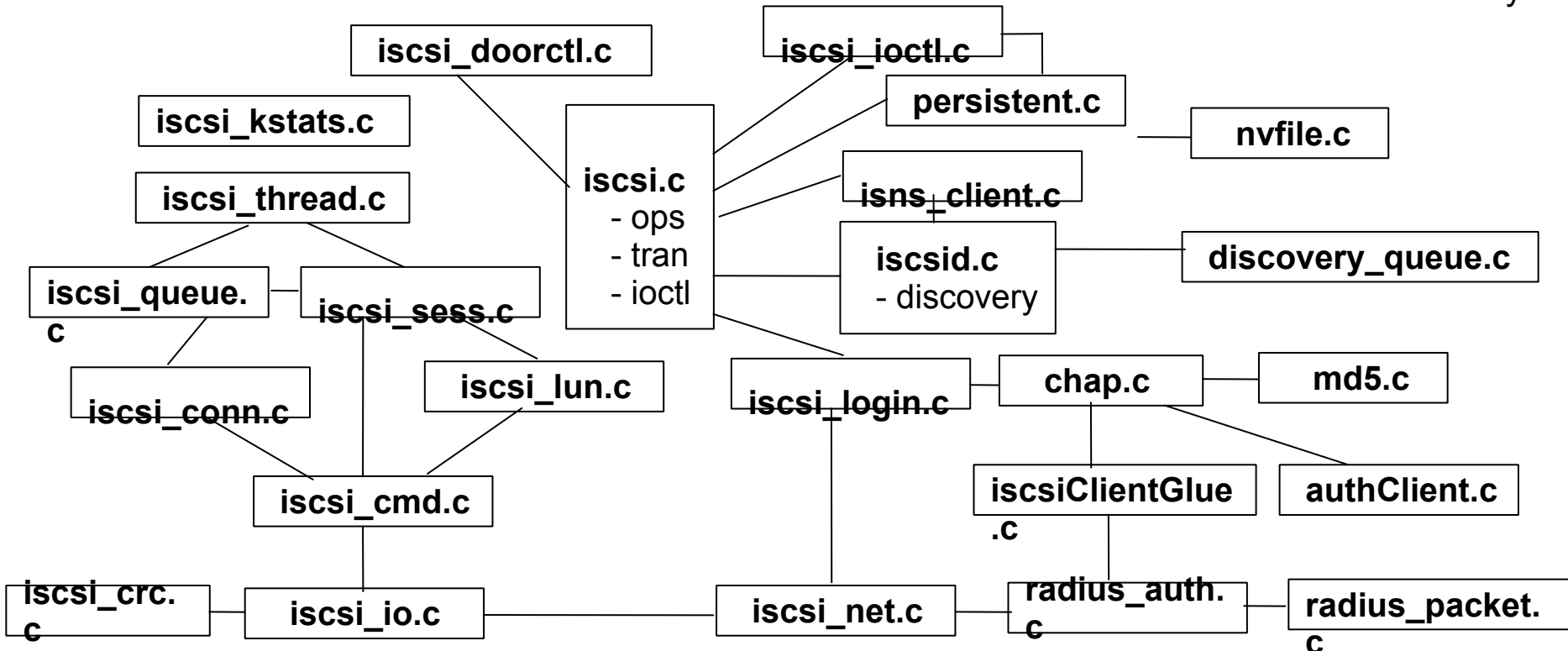
iscsi_protocol.h

Common Code

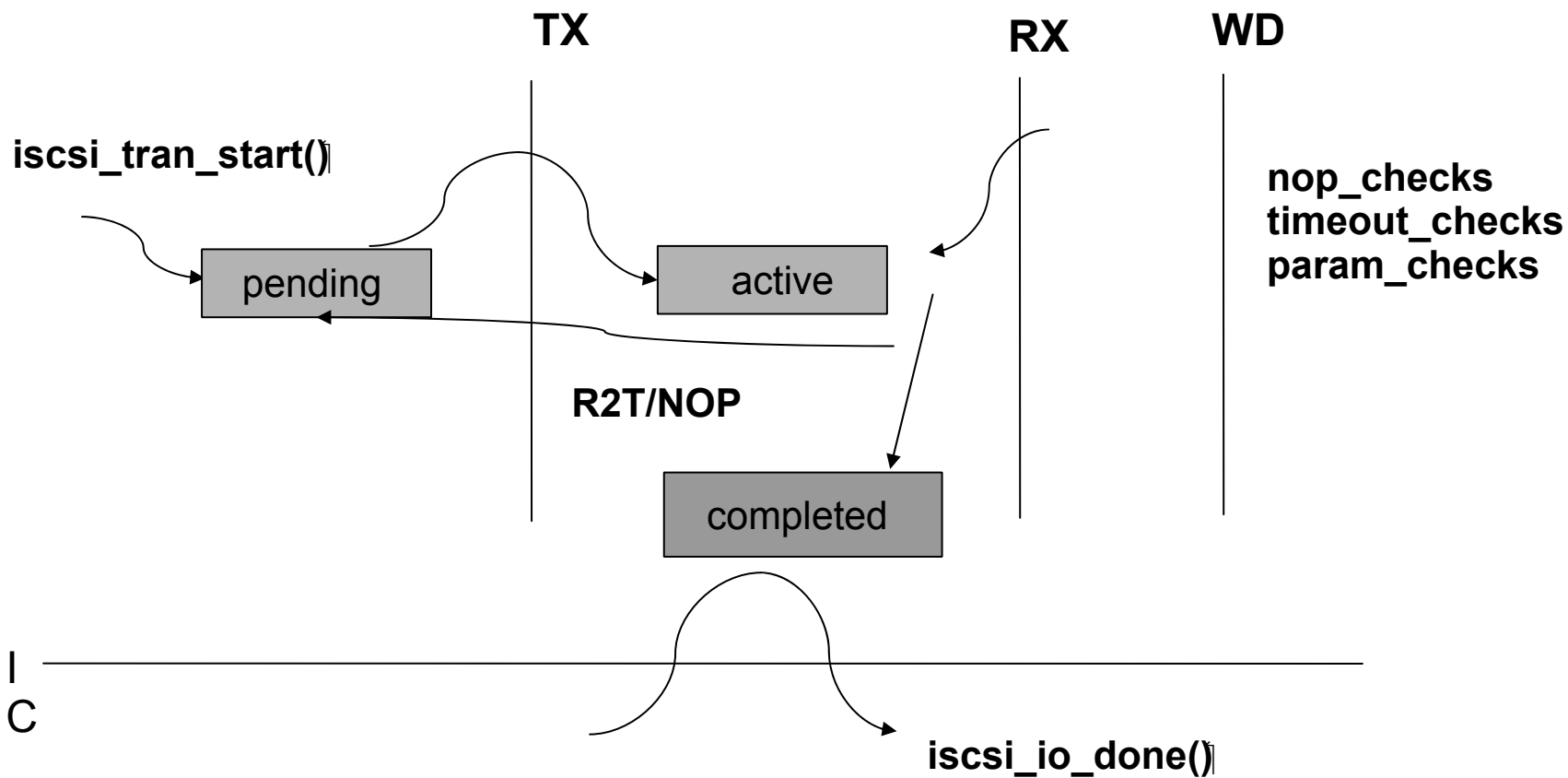
iscsiadm.c

cmdparse.c

Kernel / user land boundary



IO Model



IO Model

- **TX**
 - so send blocks, although `iscsi_tran_start` is non blocking
- **RX**
 - Always needs to listen to target
 - **ASYNC PDU**
 - **NOP PDU**
- **IC**
 - Target driver might call back into iscsi driver to a blocking call
 - `iscsi_tran_abort`, `iscsi_tran_reset`, etc
 - If we used the RX thread, we would deadlock our self
 - If we used timeouts, it would lead to deadlocks in our network stack
 - **Network stack uses timeouts**
 - **Timeouts are just tasks submitted to a common taskq**
 - **Common taskq has limited resources and submitting lots of blocking tasks is a no-no**
 - Reduces latency to use our own thread to complete IO
- **WD**
 - Used for timeouts and nop checks

State Flows



Locking flow follows the same order

cmd_mutex --> conn_mutex --> sess_mutex

Queue locking works in this order

pending_queue --> active_queue --> completion queue

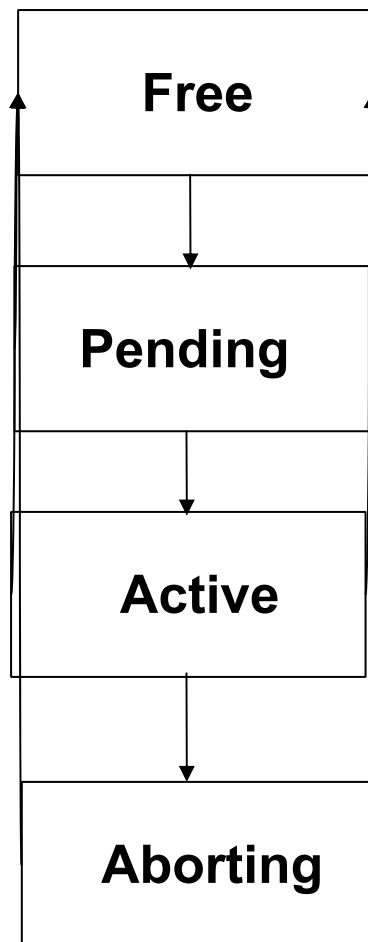
Connection and Session states are visible via kstats



Command State Flow

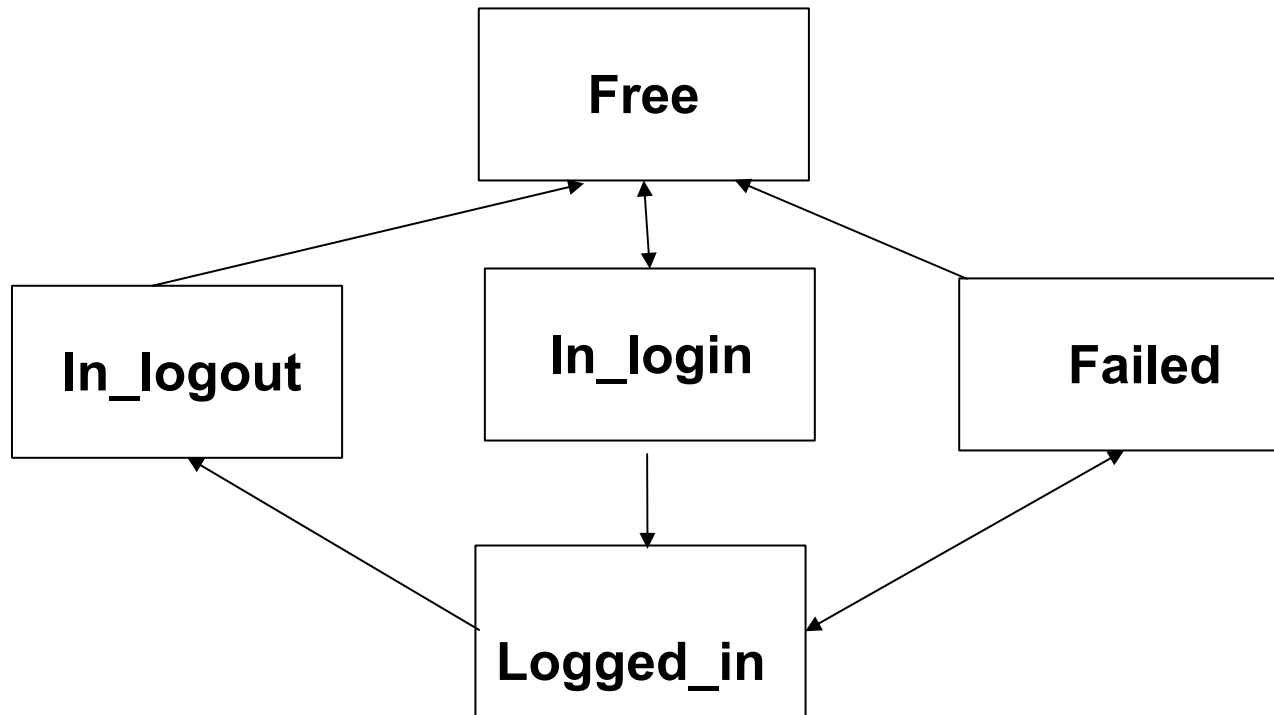
SCSI

Internal



The same diagram can be found in `iscsi_cmd.c` with lots more explanation

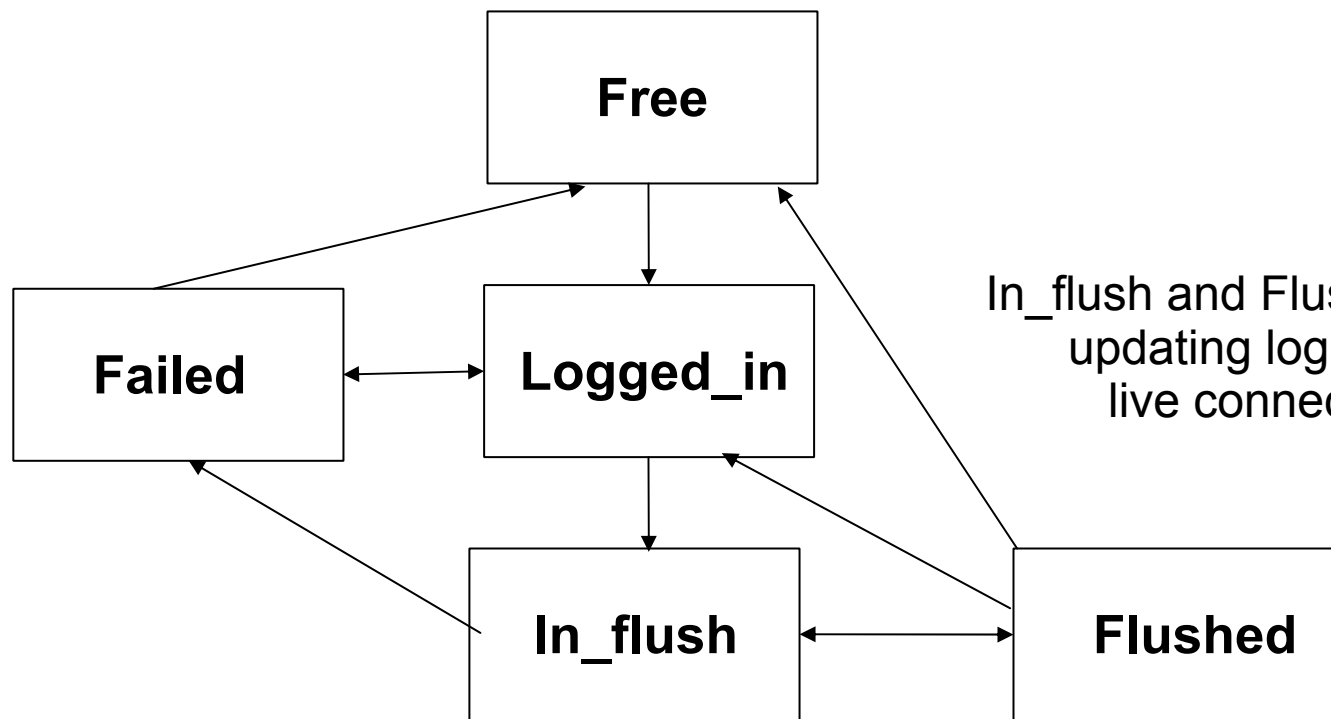
Connection State Flow



The same diagram can be found in `iscsi_conn.c` with a lot more explanations. This state diagram is based on the iSCSI specification recommendations



Session State Flow



In_flush and Flushed occur when updating login parameters on live connections / sessions.

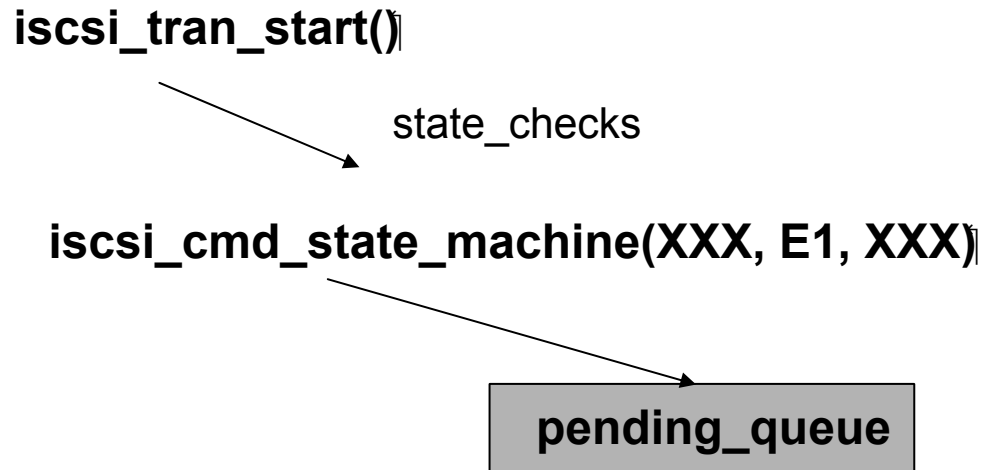
The same diagram can be found in `iscsi_sess.c` with a lot more explanations. This state diagram is based on the iscsi specification with modifications to support out parameter management

IO Code Flow

- **Four Stages**
 - E1 – command received
 - E2 – command placed on wire
 - E3 – command response from wire
 - E8 – command completed
- **Error handling**
 - E4 – command aborted
 - E6 – command timed out
 - E7 – command's connection reset

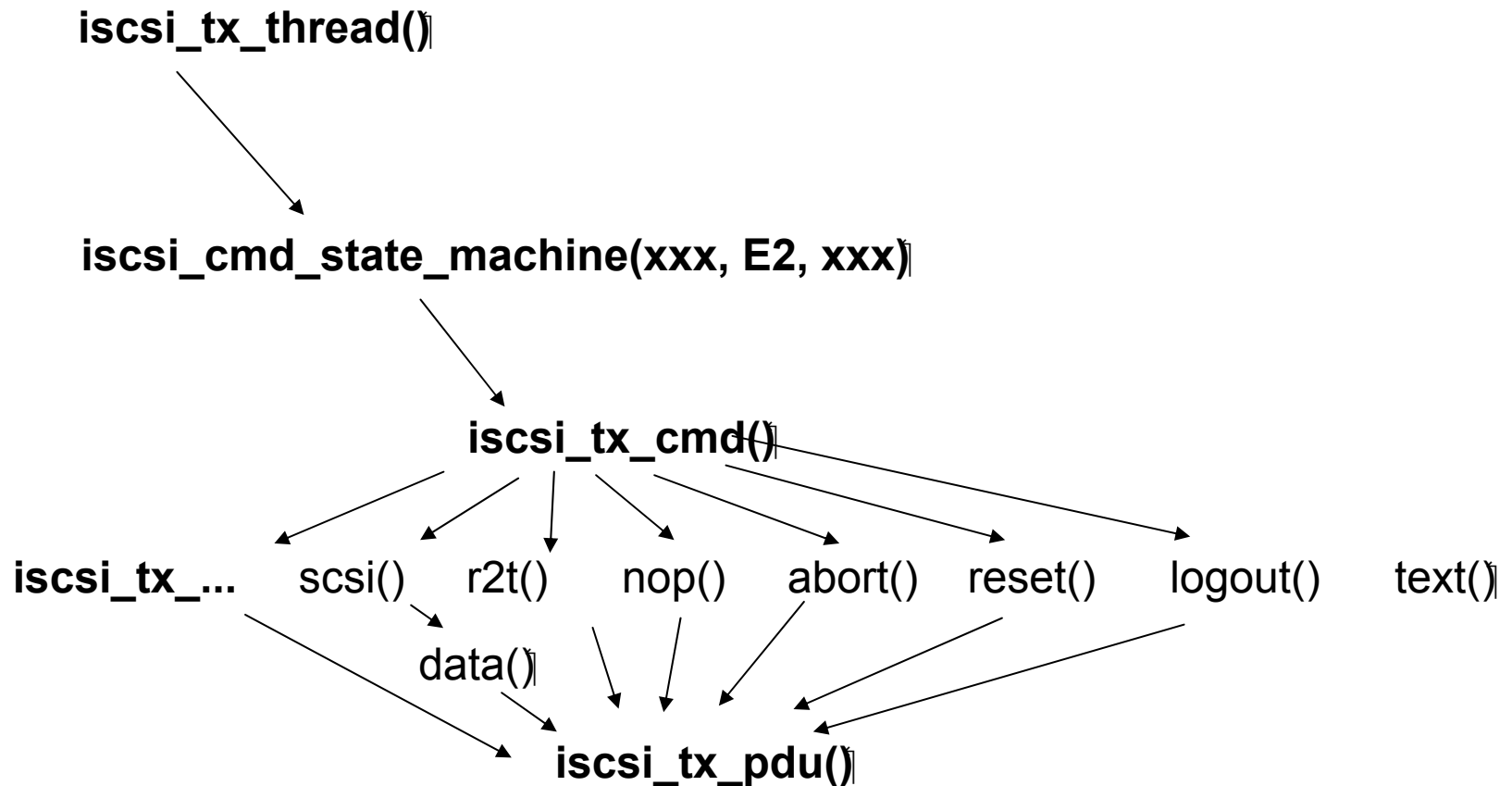
IO Code Flow

E1 – command received



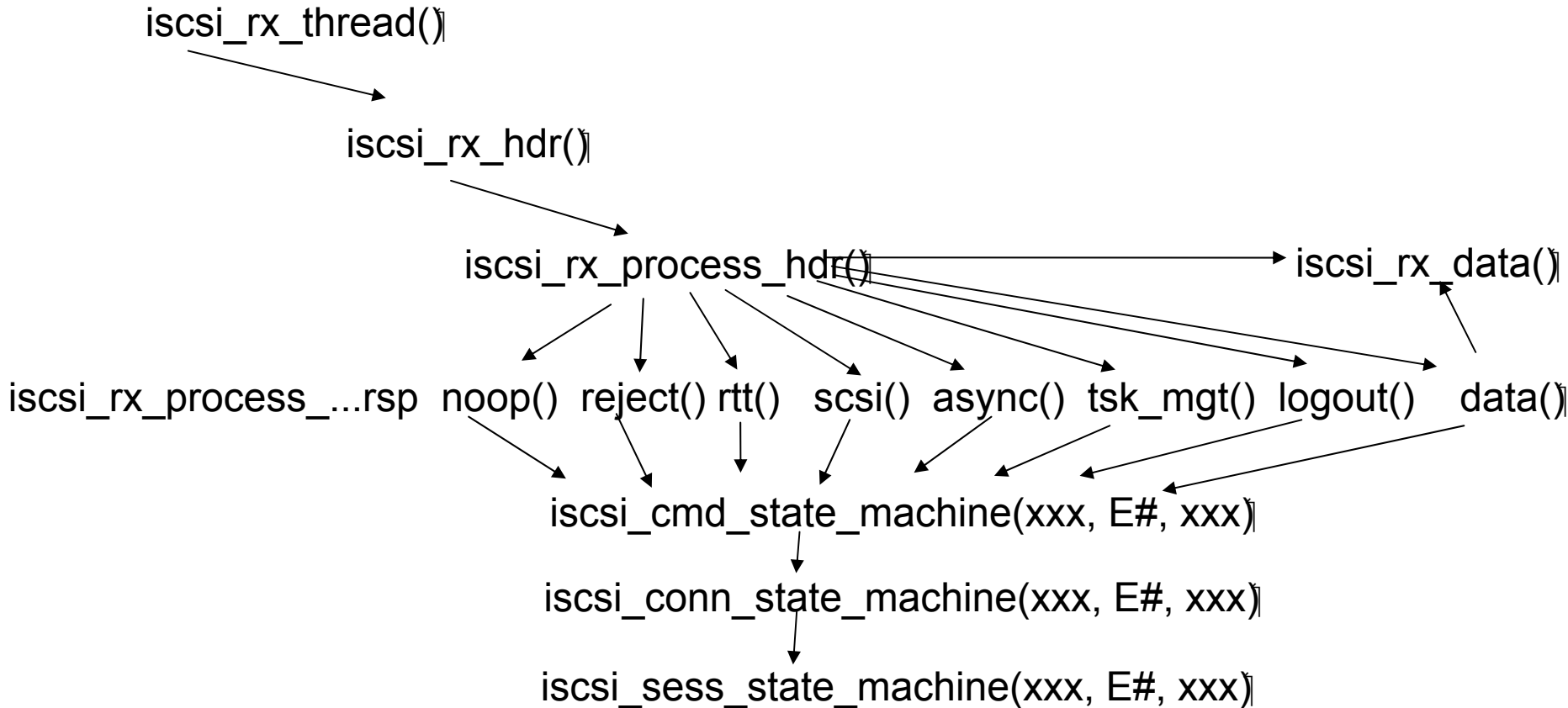
IO Code Flow

E2 – command placed on wire



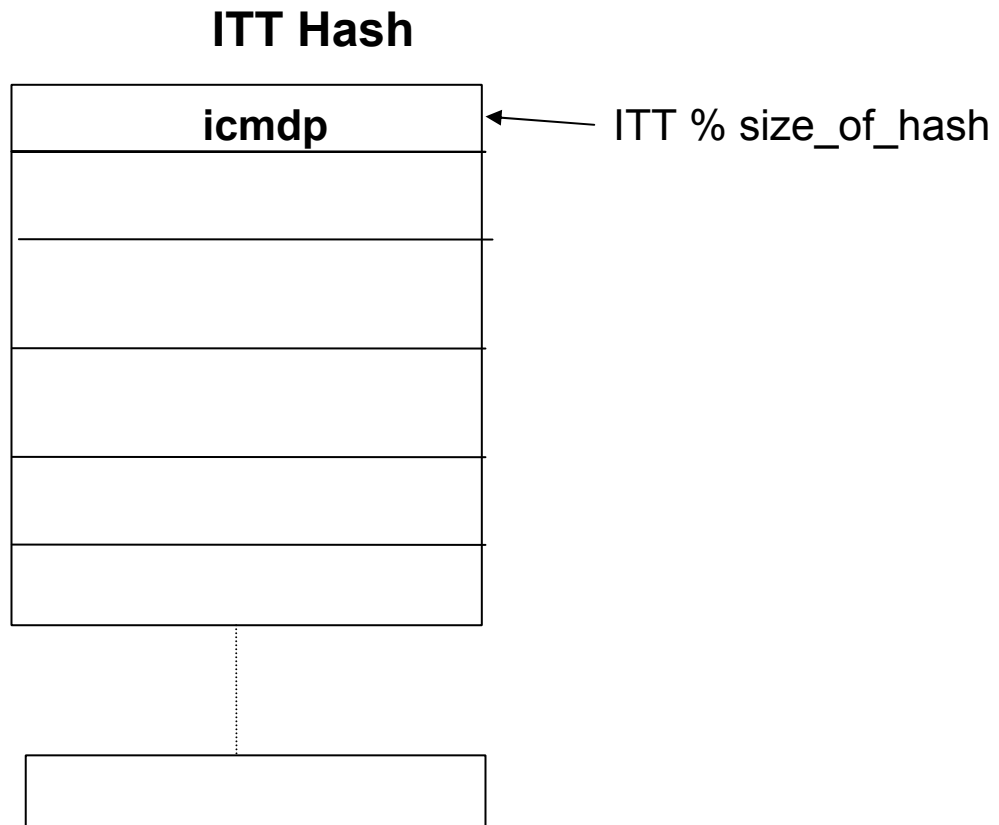
IO Code Flow

E3 – command response on wire



IO Code Flow

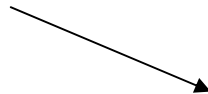
E3 – command response from wire



IO Code Flow

E8 – command completed

`iscsi_ic_thread()`



Takes command from completion queue

`iscsi_iodone()`

Discovery

- On Demand (Solaris 10 devfs)
 - iSCSI took this approach
 - iSCSI requires more resources since we have no offload. We took this approach to avoid using those resources until they are required by the OS

On Demand

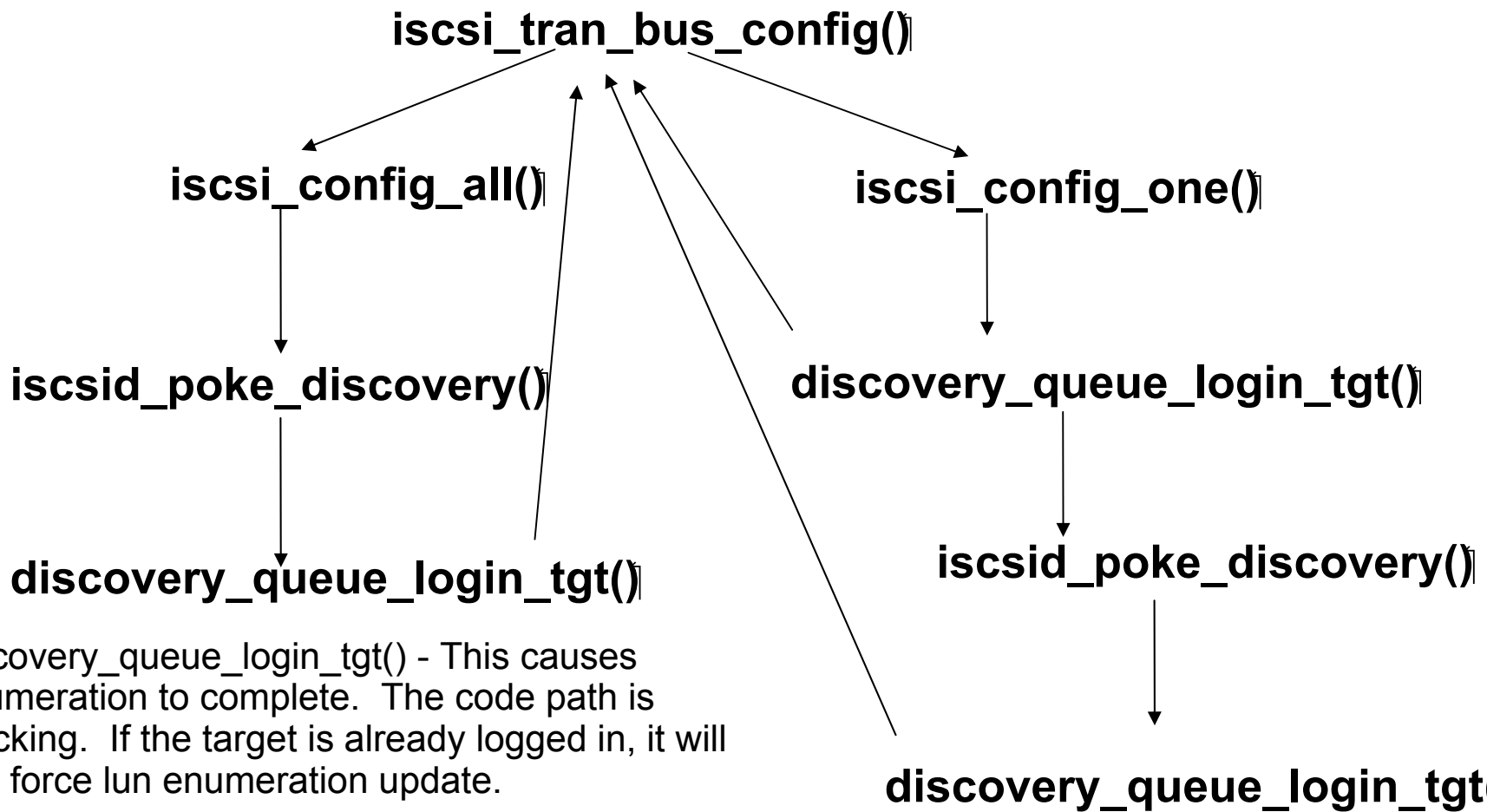
- **Entry Points**
 - tran bus_config
 - tran_bus_unconfig
- **Type**
 - BUS_CONFIG_ALL
 - BUS_CONFIG
 - BUS_CONFIG_ONE
 - **Address format for a config one**
 - [4 bytes][targetname][4bytes],[lun]
 - » First four bytes – place holder for port binding
 - » Target name (iqn.xxx-xx.com, eui.xxx)
 - » Last 4 bytes – target portal group
 - Address is generated by tran_get_name

On Demand

- **How to force a bus config**
 - **BUS_CONFIG_ALL**
 - **Is /devices/iscsi**
 - **BUS_CONFIG_ONE**
 - **Is /devices/iscsi/disk@<address>,<lun>**
 - **Address is generated by tran_get_name**
- **What does this really mean?**
 - **Initial /dev link creation is done by devfsadm**
 - **The /dev link points to a /devices/iscsi/disk@<address>.<lun>**
 - **Opening a /dev/rdisk will force a device login to occur**



On Demand



On Demand

- `iscsid_poke_discovery()`
 - Causes all discovery services that are enabled to update their caches
 - Services
 - static
 - send-targets
 - iSNS
- Unconfig is the reverse operation
 - Disabling discovery causes logout.



Enumeration

iscsi_sess_enumeration()

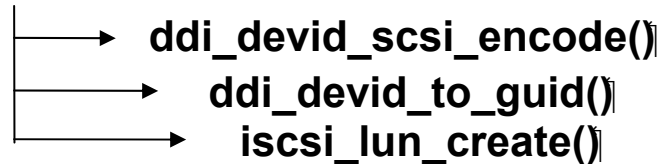


report_luns was successful.
Inquiry all LUNs found.

report_luns failed.
Inquiry LUN 0.

iscsi_sess_inquiry()

- collect page 0x00
- collect page 0x83



guid != null

iscsi_lun_virt_create()

mdi_failure



iscsi_tran_lun_init
/etc/driver_aliases
target driver attach()

iscsi_lun_phys_create()





Debugging

- **iSCSI CLI**
 - **Target list report**
 - **# iscsiadm list target.**
 - **Make sure all targets are listed with connections > 0**
 - **List discovery method**
 - **# iscsiadm list discovery**
 - **Make sure the desired discovery method is enabled**
 - **Make sure IP address specified are correct**
 - **Check /var/adm/messages for any error message**

Debugging

- **iSCSI KSTATS**

- **Show bytes we receive and transfer to the networking stack**

- **-# kstat -m iscsi 1 | grep bytes**

<u>cntr_rx_bytes</u>	<u>108567954</u>
<u>cntr_tx_bytes</u>	<u>139345954</u>

- **Show connection and session state values for each target**

- **# kstat -m iscsi | grep state**

<u>state</u>	<u>logged_in</u>
<u>state</u>	<u>logged_in</u>



Debugging

- **iSCSI Dtrace**

- **Session State Machine**

Example of logging the internal session state machine. This will create a high volume of output during login, io error, disconnect, or hang issues.

```
sdt:iscsi:iscsi_sess_state_machine:event  
{  
    printf("sess_state_machine:\t%u\t%s\t%s",  
        (((iscsi_sess_t *)arg0)->sess_oid),  
        (string)arg1, (string)arg2);  
}
```



Debugging

- iSCSI Dtrace
 - Session State Machine (continued)

Example output:

```

1 37160 iscsi_sess_state_machine:event sess_state_machine: 2 free N1
1 37160 iscsi_sess_state_machine:event sess_state_machine: 4 free N1
0 37160 iscsi_sess_state_machine:event sess_state_machine: 2 logged_in N1
0 37160 iscsi_sess_state_machine:event sess_state_machine: 4 logged_in N1
                                     ^ ^ ^
                                     ^ ^ event
                                     ^ ^
                                     ^ current state
                                     ^
                                     oid

```



Debugging

- **iSCSI Dtrace**
 - **Connection State Machine**
 - **Similar example of logging internal connection states**

```
sdt:iscsi:iscsi_sess_state_machine:event
{
    printf("conn_state_machine:\t%u\t%s\t%s",
          (((iscsi_conn_t *)arg0)->conn_oid),
          (string)arg1, (string)arg2);
}
```

Example Output:

```
1 37159 iscsi_conn_state_machine:event conn_state_machine 3 free
T1
1 37159 iscsi_conn_state_machine:event conn_state_machine 3 in_login
T1
1 37159 iscsi_conn_state_machine:event conn_state_machine 3 logged_in
T1
```

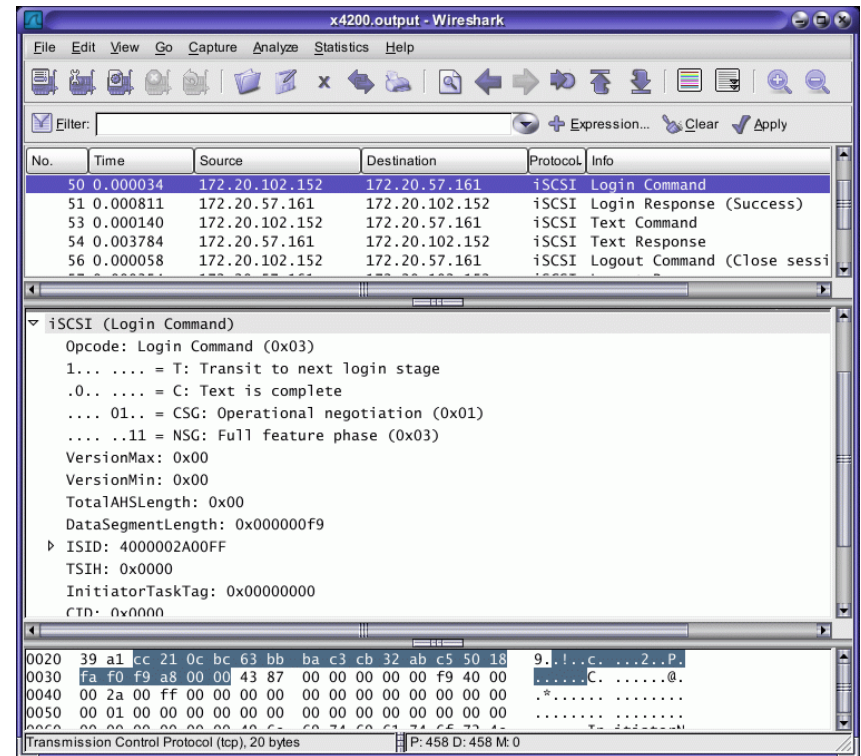
Debugging

- **iSCSI Dtrace**
 - **Force Network Disconnects**
 - **4 ways this can occur**
 - Driver has encountered an unrecoverable iSCSI protocol error
 - Target requested the disconnect
 - The user has requested the driver updates its target parameters
 - Login failures
 - **Script to detect these disconnects and why**

```
fbt:iscsi:iscsi_net_disconnect:entry
{
    printf("iscsi_net_disconnect:\t%u",
          (((iscsi_conn_t *)arg0)->conn_oid));
    stack();
}
```

Debugging

- **SNOOP/ Ethereal (Wireshark)**
 - Use Snoop to capture packets from the network
 - # snoop -o /tmp/output
Using device /dev/eri
(promiscuous mode)
14525
 - View iSCSI packets with wireshark
 - <http://wireshark.org>



Questions?