Real-Time Networking for Quality of Service on TDM based Ethernet

Badri Prasad Subramanyan
Master’s Thesis Defense
26th Jan 2005

Committee:
Dr. Douglas Niehaus
Dr. David Andrews
Dr. Jerry James
Presentation Outline

- Introduction
- Related Work
- Objectives
- Background Information
- Implementation Details
- Performance Evaluation
- Conclusion
- Future Work
Introduction

- Real-time applications span across multiple systems.
- Network delays have to be predictable to enable real-time applications.
- Present technology - Ethernet
  - CSMA/CD protocol.
  - Collisions – retransmissions.
  - Not deterministic.
  - Cannot support real-time applications.
Related Work

• Hardware Solutions
  • Shared Memory Methods
  • Switched Ethernet
  • Token Passing Protocols

• Software Solutions
  • RTnet – Hard real-time network protocol stack for RTAI.
  • Rether – Real-time Ethernet protocol.
  • Traffic Shaping
  • Master/Slave Protocols
Objectives

- Support real-time applications which span across multiple systems in a LAN.
- Predictable end-to-end packet transfer time.
- Devise a software based solution which works using the present Ethernet hardware.
- Modifications should not affect existing protocols.

Proposed Solutions
- Provide Quality of Service for real-time processes.
- Time Division Multiplexing to make the network deterministic.
- Use the framework provided by KURT-Linux.
Background Information

- UTime
- DSKI/DSUI
- Netspec
- Group Scheduling Framework
- Linux Traffic Control
- Linux Network Stack
  - Transmit packet flow
  - Receive packet flow
UTime

- Periodic interrupt at a rate of once for every 10ms in Linux 2.4.x
- Timer resolution not sufficient for real-time requirements.
- One-shot mode – interrupts the kernel at specified times.
- Timer bottom half execution in the hardirq context.
- Utime timers were used for accurate control over time.
DSKI / DSUI

- Debugging kernel code is complex
  - Kernel source code is very big and complex.
  - Large amount of concurrency
  - Kernel does not/cannot provide usage of break points in the code.

- Data Stream Kernel Interface (DSKI)
  - Instrumentation points in the kernel to log an event.
  - Interface should be configured to log events as desired.
  - Post processing filters.
  - Data Stream Visualizer.

- Data Stream User Interface (DSUI)
  - Log events in user space.
Netspec

• Provides a framework to centrally control daemons running on other systems.
• Control distributed applications.
• Transfer of files to and from daemons.
• Takes a script to specify experimental parameters.
  • Scalable
  • Reproducible
• Used to control different experiments in a LAN.
Group Scheduling Framework

- Linux scheduler is a priority-based scheduler.
- Computational Components
  - HardIrqs – executed at the earliest.
  - SoftIrqs – deferrable functions.
  - Process – scheduled by the scheduler.
- Each process has a static and dynamic priority.
- Process scheduled based on the goodness value.
- No direct control over the scheduling policy.
Group Scheduling Framework

- Group Scheduling framework provides Unified scheduling model.
- Configurable hierarchic decision structure which decides which computation to execute.
- Explicit control of computational components.
Group Scheduling Framework

- **Group** – Place holder for other entities.
- **Member** – Computational component which can be a hardirq, softirq, process or a group.
- **Scheduler** – Decision routine which chooses a component for execution.
Group Scheduling Framework

- Function pointer hooks to control scheduling and execution.
- Each of the hooks point to the Vanilla Linux routines by default.
- These hooks can be mapped to other routines to define customized routines.
- Gives the flexibility to control scheduling and execution of selective computational components.
Linux Traffic Control

- Tools for managing & manipulating the transmission of packets.
- Net_device_layer of Linux protocol stack.

Linux Traffic Control

- Components of Traffic Control
  - Queuing Discipline – a set of queues which are used to hold packets.
  - Classes – a class represents a class of packets and is associated with a queuing disciplines.
  - Filters – filters are used to classify packets.
Linux Network Stack

• Explanation can be split into 2 main parts
  • Transmit side – The code which transmits the locally generated packet out of the system. This code splits the message to be transmitted, embeds it into a packet and transmit it out of the system.
  • Receive side – The code which processes a packet received by the system. This code de-fragments the received packet and sends it to the appropriate user-level process.
  • DSKI instrumentations were used to understand the packet flow through the Linux network stack.
Transmit Packet flow

- The Application layer uses a system call to transmit a packet.
- The Transport layer (TCP/UDP) protocol stores the message in the sk_buff structure.
- The Network layer (IP) protocol does a route lookup based on the destination IP address.
- The Data Link layer transmits the packet out of the device.
Transmit Packet Flow

Application Layer
- Application writes to the Socket
- Socket writes to the INET Layer
- Socket writes to the protocol

Transport Layer (TCP/UDP)
- Transport Layer creates the packet
- Transport Layer builds Header

Network Layer (IP)
- IP receives packet & adds IP Header
- IP fragments the packet if required
- Packet is added to the device queue

Link Layer
- Scheduler checks the device queue
- Scheduler checks if the device is free
- Packet is processed by the device
- Packet goes out on the medium
- Device interrupts - transmission complete

Softirq Context

Hardirq Context
Receive Packet Flow

- The Application layer uses a system call to receive a packet. The process waits in the Transport layer queue till the packet arrives.

- The Data Link layer receives a packet.
- The Network layer (IP) protocol verifies that the packet is destined to this system.
- The Transport layer (TCP/UDP) protocol adds the packet to the Transport layer queue and schedules the process which is waiting for this packet.
Implementation Details

• Transmit side
  • Classification of packets
  • Processing of real-time packets

• Receive side
  • Classification of packets
  • Processing of real-time packets

• User Interface
  • Setting priority on Transmit side
  • Setting priority on Receive side
  • Adding/Removing real-time processes
Linux Softirqs

- **Softirqs in Linux 2.4.x**
  - **HI_SOFTIRQ** – processes the high priority bottom halves and tasklets.
  - **NET_TX_SOFTIRQ** – transmits a packet out of the system.
  - **NET_RX_SOFTIRQ** – processes a packet which is received on the system.
  - **TASKLET_SOFTIRQ** – processes the low priority tasklets.
Classification on Transmit side

• Packet classification was done in the Linux Traffic Control.
• Connection or flow was identified uniquely using port numbers.
• A new queuing discipline – TDM queue was developed.
  • Contains a single FIFO queue by default.
  • Can be configured to contain more than one queue.
  • Each queue is associated with a particular priority.
  • Each queue enqueues packets pertaining to a connection.
TDM queue

- Packets are enqueued in different queues based on their priority.
- Dequeueuing of packet starts from the queue with the highest priority.
- The default queue is the last queue to be processed.
Group Scheduling Framework - TDM

- Timer_BH is processed first.
- NET_TX_SOFTIRQ has the highest priority during transmission time slot.
- Other softirqs are processed sequentially.
- NET_KFREE_SKB_SOFTIRQ frees the packet after transmission.
Receive side

- Classification of packets.
- Reduce the real-time packet processing time.
  - Processing the real-time packets before the non real-time packets.
  - Reduce the wait time for a real-time packets in the Transport layer queue.
Classification on Receive side

- Packet classification was done using a queuing discipline on the receive side.
- Connection or flow was identified using the port numbers.
- The queuing discipline on receive side
  - Contains a single FIFO queue by default.
  - Can be configured to contain more than one queue.
  - Each queue is associated with a particular priority.
  - Each queue enqueues packets pertaining to a connection.
Modifications to Receive Softirq

- **NET_RX_SOFTIRQ** was split into 3 parts
  - **NET_RX_SOFTIRQ** – responsible for classifying a packet and enqueuing it appropriate queue.
  - **NET_RX_PRIORITY_PROCESS_SOFTIRQ** – responsible for processing the priority packets present in the priority queues.
  - **NET_RX_NORMAL_PROCESS_SOFTIRQ** – responsible for processing the non-priority packets present in the default queue.
Queue on Receive Side

- Packets are enqueued in different queues based on their priority.
- Dequeueuing takes place based on the priority of the queue – NET_RX_PRIORITY_PRIORITY_PROCESS_SOFTIRQ.
- The default queue is scheduled to be processed after all the priority queues are processed – NET_RX_NORMAL_NORMAL_PROCESS_SOFTIRQ.
Priority Group

- Reduce the wait time in the Transport layer queue.
- Addition of Priority Group to the Group Scheduling Framework.
- Contains all real-time processes.
- A process in the ‘Priority’ group is scheduled once the packet corresponding to this process reaches the transport layer queue.
Real-Time Networking Model

Top Group

- TIMER_BH
- NET_RX_SOFTIRQ
- NET_RX_PRIORITY_PROCESS_SOFTIRQ

TDM Group
- NET_TX_SOFTIRQ

Softirq Group
- HI_SOFTIRQ
- NET_TX_SOFTIRQ
- NET_RX_SOFTIRQ
- NET_RX_NORMAL_PROCESS_SOFTIRQ
- NET_KFREE_SKB_SOFTIRQ
- TASKLET_SOFTIRQ

Priority Group
- Process P1,...,Pn
User Interface

- User interface in TDM
  - /dev/tdm_controller – pseudo device.
  - The loadable module to interface with the device.
- This module was enhanced to configure the device for QoS changes.
- User-level program with command line options.
  - Set priority on the transmit side.
  - Set priority on the receive side.
  - Add/remove processes from the Priority group.
Setting Priority on Transmit/Receive side

- Command:
  
  tdm send <# of queues> <list of (port# priority)>
  tdm recv <# of queues> <list of (port# priority)>

- tdm – user-level routine.
- send/recv – set the transmit/receive queues.
- # of queues - Number of queues to create on the transmit side.
- Port# - the port number which needs to be prioritized.
- Priority – the priority of the port number. (Smaller number has higher priority)
Add/Remove prioritized process

- Command:
  
  tdm add process <list of process Ids>
  tdm remove process <list of member Ids>

- Tdm – user-level routine.
- Add process – adds a process to the ‘Priority’ group.
- Remove process – removes a process from the ‘Priority’ group.
- Process ID – Unique identifier for each process on Linux.
- Member ID – Unique identifier for a Group Scheduling member.
Performance Evaluation

- End-to-End packet transfer time for a single real-time process.
- End-to-End packet transfer time for a single real-time process with background processes.
- End-to-End packet transfer time for different TDM schedules.
- Packet processing time on the transmit and receive side of the network stack.
- Visualization of the output.
Test Setup

- LAN with 4 systems connected using a 100 Mbps hub.
- Time synchronization master.
- Transmission slot time – 260s (220 s transmission time + 40 s buffer time).
- Total transmission cycle of 1040 s.
- 64 bytes of data.
End-to-End Packet Transfer Time

- Single real-time application.
End-to-End Packet Transfer Time

- Real-time/non real-time process with other background processes.
Packet Processing Time

- Packet processing time in the kernel.
Pipeline Computation

- 3-stage pipeline computation.
- LAN with 4 systems using a 100 Mbps hub.
- Time synchronization master.
- Transmission slot time – 260s (220 s transmission time + 40 s buffer time)
- Total transmission cycle of 1040 s.
Pipeline Computation Visualization

Packet processing on the transmit side.
Packet propagation time.
Packet processing on the receive side.
Pipeline Computation

- Pipeline computation gives a real-world distributed application scenario.
- The packet processing time on the transmit side varies depending on the availability of the transmission time slot.
- The packet processing time on the receive side does not vary much.
- The end-to-end packet transfer time is comparable to half total transmission time cycle.

<table>
<thead>
<tr>
<th>Table 5.2: Packet processing time</th>
<th>Average Time (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet processing time on Transmit side</td>
<td>492</td>
</tr>
<tr>
<td>Packet processing time on receive side</td>
<td>41</td>
</tr>
<tr>
<td>End-to-End transfer time</td>
<td>537</td>
</tr>
</tbody>
</table>
Conclusion

- Differentiation of real-time and non real-time packets in the network stack was achieved.
- Predictable end-to-end transfer time was achieved in a LAN.
- This software solutions was implemented on the standard Ethernet hardware.
- The changes do not require any modifications to the protocols used in Linux.
Future Work

• **Time Constraint in Quality of Service**
  - Each process has a set of QoS parameters.
  - Each process should negotiate QoS with the system based on its QoS parameters.
  - System should commit to provide desired QoS only when it has the resources.

• **Resource Reservation Protocol (RSVP)**
  - Extend QoS beyond a single LAN.
  - RSVP reserves resources in the intermediate nodes for an Internet connection.
  - The real-time networking module can interact with the RSVP protocol to reserve resources on nodes beyond the LAN.
Thank you