Handling Failures and DOS Attacks Using Network Device Groups

Ramkumar Chinchani, Suranjan Pramanik, Ashish Garg
Dept. of Computer Science and Engineering
University at Buffalo, SUNY
Buffalo, NY 14260
Email: {rc27, pramanik, ashish}@cse.buffalo.edu

Abstract

With the growing popularity of the Internet and the falling prices of network devices, it is not unusual to find multiple network devices in a computer system. Technologies such as Internet connection sharing and NAT are commonly being used by end users to make network connectivity more viable. In this paper, we point out that this implicit redundancy can be used to achieve fault tolerance. It is known that network devices can be grouped to achieve failover support. However, the focus has been limited to localized factors and device failures. In the context of the Internet, security against DOS attacks also becomes an important issue. While the use of multiple network devices provides a good solution for device failure, it doesn't guarantee a good defense against DOS attacks. We show that computer systems can become tolerant to DOS attacks if some external factors are also taken into account. The main contribution of this paper is a systematic and comprehensive solution that makes a best effort to provide reliable network connectivity even when network device failures and DOS attacks occur. We have implemented and tested this technique in Linux and report our findings.

1 Introduction and Motivation

Network connectivity is becoming widely available, making it possible to move a large share of communications and transactions to the Internet. The required infrastructure is provided by a smorgasbord of network devices and protocols interacting with each other. Like any software or hardware component, there is always a possibility of failure at any point. The problem of providing reliability and availability is solved in different ways at the software and hardware level. Some fault-tolerance is already built into protocols such as TCP that attempt to provide reliable communication over unreliable networks. Redundancy and replication is a

common theme of most solutions addressing the problem of fault-tolerance at the hardware level. However, on-board replication is not employed in commonly used network hardware such as PCI or PCMCIA cards because the costs become prohibitive. Instead, multiple cards are used to provide failover support. Failures can also happen due to deliberate attacks such as denial-of-service (DOS) attacks. The idea behind the attack is to overwhelm the target with a large number of service requests with the goal of disrupting availability. Although this may be construed as a transient failure, the downtime could prove expensive in terms of lost transactions and business.

The basic concept of failover switching is not new and it has found applications in the domain of highly available database and computing clusters [1, 2]. The main focus of this paper is to provide a systematic and comprehensive solution to address the issue of continued network connectivity in the face of both network device failure and DOS attacks. Any discussion of failure in this paper corresponds to both of these failure events unless explicitly stated. Some of the outstanding goals and motivation of this work are as follows:

- Systematic analysis

  Merely the availability of two or more devices connected to the network does not ensure fault-tolerant connectivity. For example, let a computer system have two network devices, each connected independently to a hub. If one network device is under a DOS attack through the hub, then switchover to another device does not provide a solution since the hub itself is being overwhelmed by the attack. A similar argument can be made when the hub experiences a hardware failure. Although faults due to device failures and attacks may resemble each other, there are some fundamental differences in the context of achieving reliable connectivity. We defer the elaboration of these differences to later sections.

- Rapid and transparent failover switch

  Failover switching is a response to a failure. Therefore, quickly establishing that a failure has occurred is important. If a protocol, such as “heartbeats” [8], for failure detection is used on the top of existing network protocols, then these intermediate layers may introduce latency and interfere with rapid failure detection. Instead, we argue that failure detection should be performed at the frontlines of network communication on a host, i.e., at the network device level. Once it is known that a device failure has occurred, a failover switch that is performed should be transparent and attempt to minimize any packet loss.

- A cost-effective solution

It is not unusual to find a computer system with multiple network devices, each device connected to the Internet via different means. For example, ethernet cards use wired technology and the 802.11 based cards use the wireless technology. Multiple devices are generally installed on a computer system for the convenience of connectivity. We propose to utilize this implicit redundancy to provide cost-effective fault-tolerant network connectivity targeted at end users.

In this paper, we devise a technique that makes a best effort to achieve continued network connectivity in the event of faults, i.e., both device failures and DOS attacks. We address the problem by taking into account local factors at the network device level as well external factors at the TCP/IP protocol and network topology level. We then perform a feasibility study of our technique using extensive OPNET [5] simulations. Following promising results from this study, we have a developed a preliminary prototype in Linux. This prototype can be downloaded from http://netdevgrp.sf.net.

1.1 Paper Organization

Section 2 compares our work with existing literature to put our work in perspective. Systematic analysis and the basic technique are discussed in Section 3. We report the results that we have obtained from simulations in OPNET in Section 4. The prototype implementation and architecture are elaborated in Section 5. We summarize our technique and discuss the future directions of research in Section 6.

2 Background and Related Work

The fault-tolerance domain is rich in techniques to tackle the problem of reliability and availability even in the presence of device or systemic failures. In particular, at the network device level, redundancy has been used to rapidly respond to device failures. Intel's Adapter Fault Tolerance (AFT) technology [3] provides continuous network connectivity to high-end servers. It uses an intelligent software agent to monitor the network links for the server. In case, one of the network link fails, the redundant link takes over immediately. There is no interruption in service. The aggregation of network adapters is called as *teaming* in this approach which is similar to the notion of *network device grouping* proposed in our technique. SCO MDI driver [7] allows networking traffic to be shifted to a different configured network adapter card when a hardware failure is detected on the original card. The failover adapter is configured by a *Network Configuration Manager*. This configuration manager also provides the ability to force a failover and to failback to the original adapter after the problem is resolved. This approach is very simplistic and is concerned with only host level issues,

limiting its effectiveness.

Failover switching has also been implemented at higher levels. IP Address Takeover (IPAT) is a technique where two hosts providing some service are hidden behind a "floating" IP address, which is visible to client hosts. This floating address is initially bound to an interface on one of the hosts. In the event of a software level failure, there is switchover to the other host. ARP packets are sent or received to update the neighbors cache to reflect the change. TCP is a connection-oriented protocol that makes a best effort to provide reliable service over unreliable media. The protocol maintains sequence numbers for each packet to track the state of the connection. However, if the interface to which the connection is bound fails, then the connection is eventually lost. The Stream Control Transmission Protocol (SCTP) [6] was proposed at the transport layer to provide greater connection reliability given multi-homed hosts.

Fault-tolerance is also used at application level. NetFORCE Enterprise-Class Information Management System [4] provides active clustered failover that eliminates single point failures. The architecture of NetFORCE 4200C relies on a dedicated heartbeat interface between the server modules which trigger a transparent online failover of all the network interfaces in the event of a server module failure.

Denial of service (DOS) [10] attacks are annoyingly simple to launch but cause significant damage by affecting service availability. Several network protocol level techniques [12], [11] have been proposed to combat this scourge.

Some goals of the above techniques are similar to our approach such as continued network connectivity in the event of device failures and attacks. However, most of the techniques either consider localized factors or propose security measures at intermediate nodes/routers. In contrast, we provide a solution to handle both device failures and DOS attacks by utilizing the redundancy provided by multiple network adapters at the end hosts. The efficacy of the technique is greatly improved if external factors such as network topology and reachability are also taken into consideration.

3 Overall Technique

Our technique assumes a multi-homed computer system with each physical network interface connected to a subnet. It is possible to define virtual interfaces bound to the same physical device, but if this device fails, then all the virtual interfaces fail. Therefore, true redundancy is available only between interfaces that are bound to different physical devices. Apart from this setup, no additional hardware is required.

The typical use of a multi-homed computer system is to serve some specific purpose such as routers or NAT boxes. They perform an important function by acting as gateways in small organizations. Computer

systems used by end users may also contain multiple network devices. However, only one may be used at any given time and others may lie quiescent. The discussion in this paper enumerates some advantages in connecting both devices if possible.

3.1 Basic Architecture

Consider a host or computer system with two or more physical network interfaces. A *network device group* is a subset of these interfaces such that each group contains two or more devices. For example, for a host with only two network interfaces installed, only one group is possible, i.e., one that contains both these devices. Under normal circumstances, each participating network device is oblivious to the grouping and the device functions are not affected in any way.

An event of interest is either a device failure, or a sudden and subsequently prolonged increase in traffic, typical of a DOS attack. A device failure is detected when errors occur during frame arrival or transmission, and the error code is return in the status word register of the device. A DOS attack is detected when frame arrival rate is close to the maximum capacity of the network device and the frame arrival rate sustains at this level for at least some preset interval of time that is set by the system administrator. Since the network device drivers are in the closest proximity to the actual event, any action that is taken is rapid. This is in contrast to the higher level protocols that ensure connection failover.

When any of these events occurs, another device in the same group is chosen as a surrogate and all the functions are transparently transferred to this device. The routers on the new path are updated by sending an ARP packet prior to sending the frames. Consequently, upper layers in the network stack are unaware of this switch. When the device that generated the original event recovers, either the functions can restored to the original device or retained with the surrogate device. Each option has its pros and cons. After a switchover has occurred, the surrogate device's frame transmission and arrival queue is effectively cut in half for the connections originally bound to the surrogate device. This is undesirable since the chances of dropped packets are increased. Assuming that we choose the option of restoring the functions to the original device, then frequent occurrence of failures causes a large ARP traffic to be sent on both links. This situation essentially calls for a trade-off depending of what the situation may be.

3.2 Criteria for Device Grouping

Creation of logical device groups generally increases the redundancy and hence improves fault tolerance both at hardware and software level. However, in the context of network devices, this is not always the

case. Not all network devices can be arbitrarily combined into a network device group. If a device in the group should take over as a surrogate for another device, then all the destinations over the network that were reachable from the faulty device should also be reachable from this surrogate device. This is a very important criterion for choosing network devices to form a group.

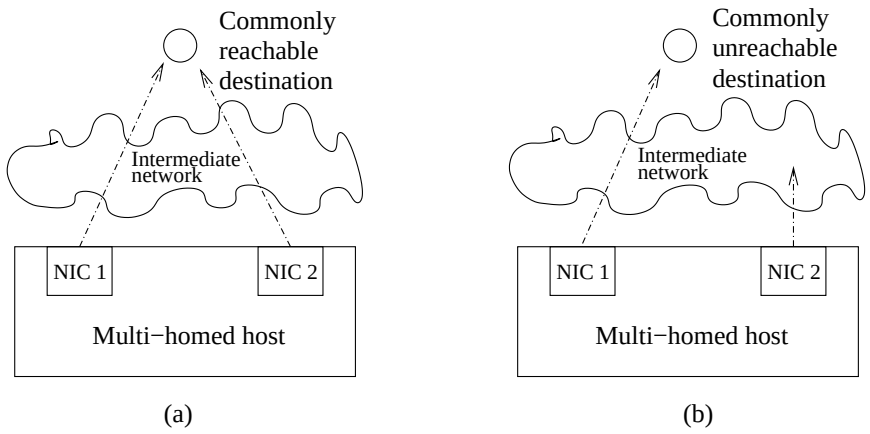


Figure 1: (a) A good, and (b) a bad network device grouping to handle device failures

Fig. 1 shows a good and bad network device grouping for a multi-homed host with two network interface cards. When a connection is made through one interface, a transparent switch can be made only if the destination can be reached via another interface as shown by by Fig. 1 (a). It is possible to achieve this by finding a common point, typically a router, that is reachable by both the interfaces. Packets with a destination address that arrive at the router through one line are now seen on another line. An ARP transmission prior to packet transmission ensures that the routing tables are updated. It is desirable if the update occurs rapidly, and for this reason, a common point must be found very close to both interfaces. If a particular destination cannot be reached through another device in the group because a common point is not available, then forming a group with these devices is not useful. This is illustrated by Fig. 1 (b).

In case of DOS attacks, the criteria for finding the common point is slightly different. DOS attacks typically involve flooding a target host with an excessive amount of traffic with the intention of overwhelming the host. If a group is formed with two interfaces in such a way that a common point exists very close to these two interfaces, then it is very likely that that common point is relaying the attack traffic and is by itself being overwhelmed (see Fig. 2 (a)). Therefore, if the second device takes over as a surrogate for the interface under attack, then the packets that are sent do not have any improved chances of being delivered. Therefore, in contrast to device failure, it is desirable to find a common point further away from the network

device group since it is more likely that independent paths exist from the two devices to the destination and a DOS attack on one path does not affect the other.

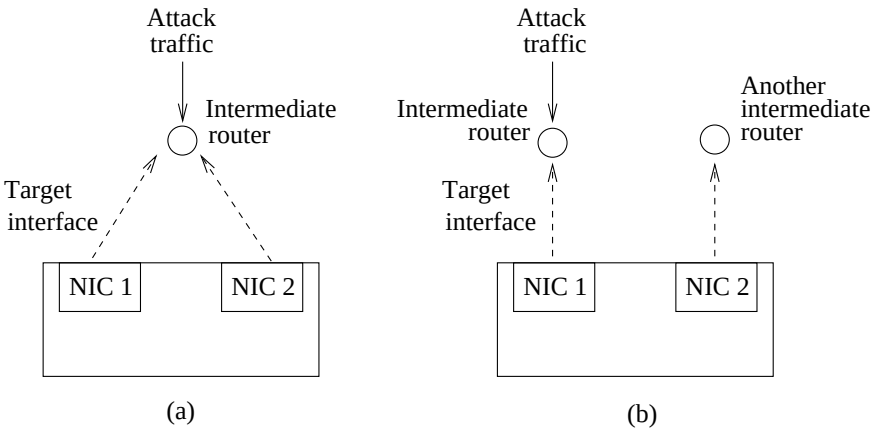


Figure 2: (a) A bad, and (b) a good network device grouping to handle DOS attacks

4 Simulations

The notion of using external factors for improved fault tolerance in the event of failures and attacks was verified by setting up a OPNET simulation. Fig. 3 shows the topology that we have considered for device failures. Important players in the setup are the multi-homed host running a HTTP server and a client host on the LAN making sporadic downloads resulting in heavy bursty traffic. The multi-homed host (see. Fig. 4) is implemented by adding another MAC node with a receiver and transmitter. The concept of network device grouping is implemented over the two MAC nodes. It arbitrates any failure events and the failover switch. We have considered two different scenarios: (1) device failure events, and (2) DOS attacks.

4.1 Device Failures

During the first simulation run, we force only one interface to fail. As a result, the other interface takes over as a surrogate device and they both remain in that state. Sending ARP packets to notify the change is important, else the host at the next hop is unaware of the change and continues to send packets on the original link, resulting in massive packet loss. Fig. 5 shows that at approximately 2 mins, one of the interfaces fails, causing a switchover. At this instant of time, there is a burst of traffic in progress on Link 0 of the multi-homed host. As soon as the switchover happens, a fraction of the burst is immediately transferred on to Link

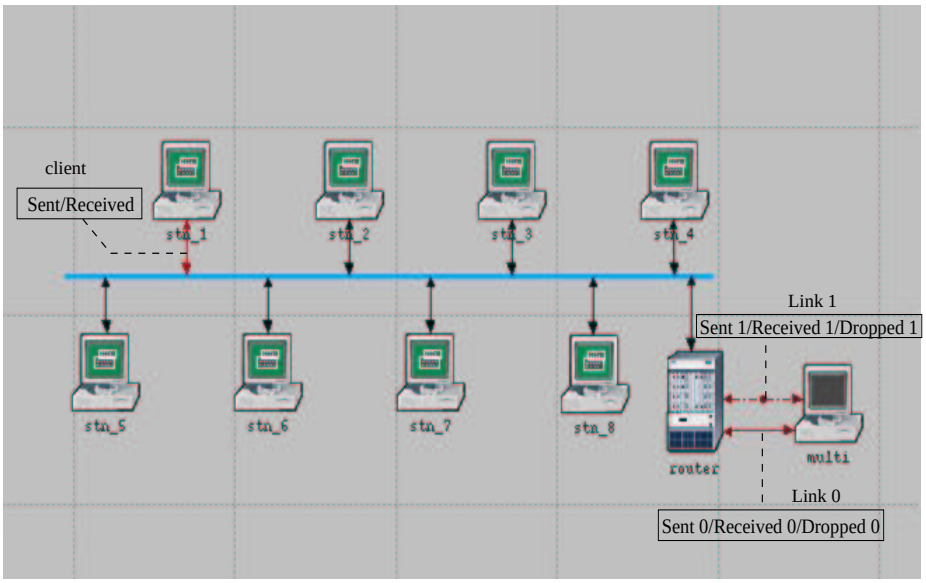


Figure 3: A network topology to model failures

1. This demonstrates a rapid switchover mechanism. Any subsequent traffic is seen only on Link 1.

During the second simulation run, we cause both interfaces to fail but with different probability distributions. Fig. 6 presents the statistics corresponding to this simulation run. At approximately 2 mins, a burst of traffic is seen on Link 0 and then a failure occurs on one interface. A switchover happens immediately, but some packets that are in transit and in the device queues are dropped. At approximately 32 mins, another burst is seen, but the data transfer happens on Link 1. This time not many packets are dropped. Also notice the stream of ARP packets that have to be sent on both links corresponding to failure events. Frequent switchovers cause a large amount of ARP traffic.

4.2 DOS Attacks

To simulate a DOS attack, we have modeled a larger network (see Fig. 7). A multi-homed host is connected to two routers and these routers are connected to the client station through a network cloud. A large amount of traffic is pumped through Link 0. The maximum capacity of the network interface is set at 0.2 MB/s. Fig. 8 shows relevant statistics on Links 0 and 1. After the traffic reaches the maximum capacity and sustains for 1 min, a switch is performed. As soon as the traffic rate reduces below 0.2 MB/s, a switch is made back to the original interface. The DOS attack that was simulated did not sustain at the maximum rate, instead it fluctuated. It can be observed that the client receives packets with very little packet loss. If all traffic was

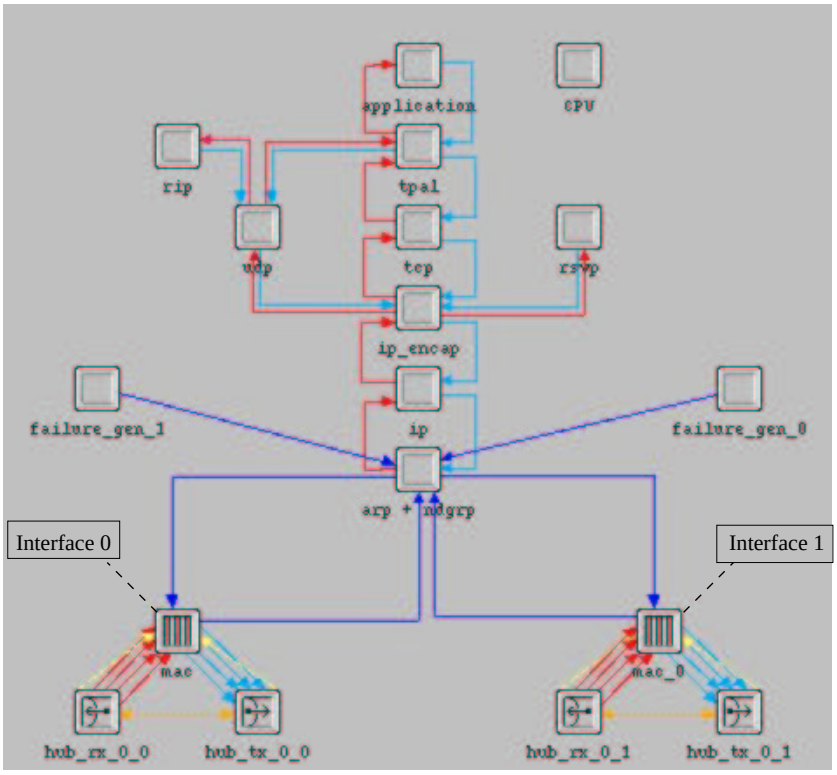


Figure 4: A node model in OPNET to simulate a multi-homed host

sent on Link 0 when the maximum capacity has been reached, there was a significant packet loss rate. This corroborates our hypothesis that a grouping of network devices based on network topology improves the chances of providing reliable network connectivity.

5 Implementation

We have implemented the network device grouping inside the Linux kernel (2.4.19). The entire implementation is divided into two parts: (1) kernel code that responds to the alerts generated by the device driver, and (2) user space code for administration of the network device groupings.

5.1 Kernel Code

The Linux kernel is organized in an object-oriented fashion. The device driver directly interacts with the network device such as a PCI ethernet card or a wireless card. The TCP/IP stack is implemented over this driver. User space applications interact with the network stack through system calls, viz., *socket(2)*, *read(2)*

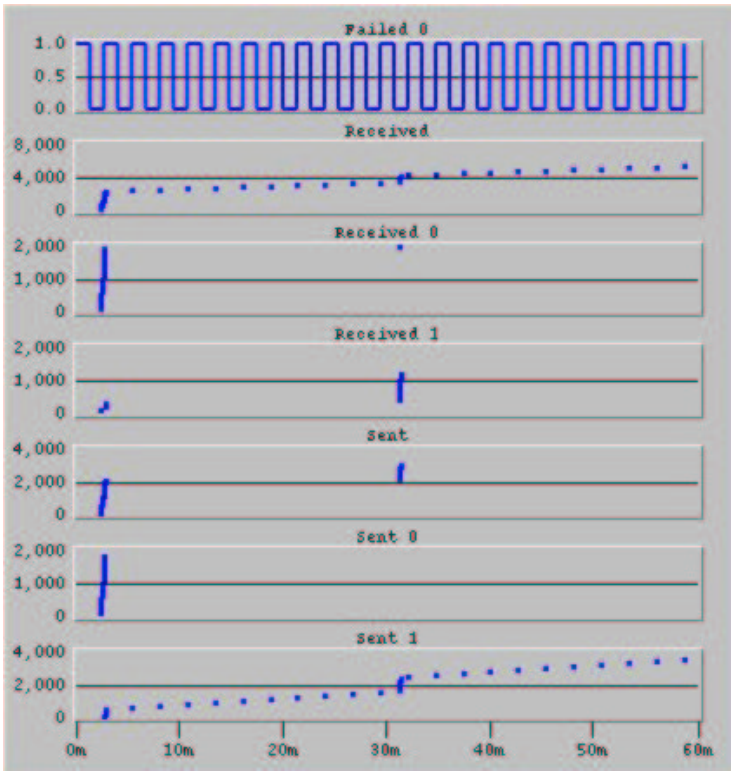


Figure 5: Relevant network statistics when only one interface on the multi-homed host fails

and *write(2)*. In order to achieve rapid failure detection and transparent failover switching, the network device grouping subsystem is implemented directly over the network device driver code. The implemented subsystem is called *netdevgrp*. A logical view of the netdevgrp subsystem and its place inside the Linux kernel code is shown in Fig. 9. Note that the netdevgrp subsystem does not directly lie in the normal data flow path of TCP/IP communications. Therefore, there is little computational overhead.

The basic structure of a device driver and low-level routines [9] is illustrated in Fig. 10. A network device is normally assigned an interrupt number and it is activated whenever an interrupt occurs. A device driver for this network device registers an interrupt handler to handle these interrupts. Each network device driver also maintains two queues, one for receiving frames from the physical media and the other for receiving frames for transmission from upper layers in the protocol stack. The device driver exports a function named *hard_start_xmit()* to the upper layers through a function pointer. Therefore, if a frame has to be delivered to the physical media, the upper layers invoke this function. Similarly, the upper layers export a function named *netif_rx()* to the device drivers. When a frame is received, it is processed to remove the headers and this function is called. This is a basic description of the interaction between various relevant components

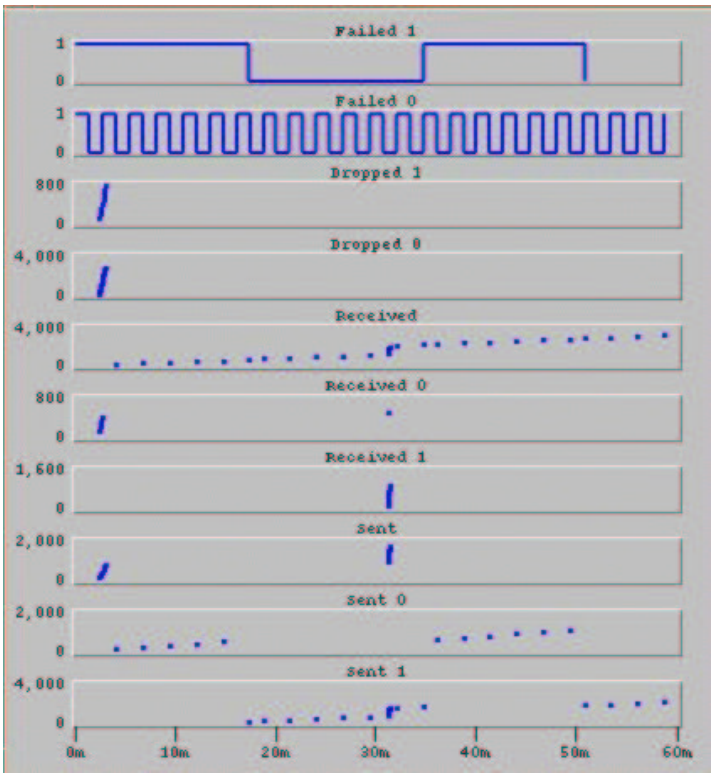


Figure 6: Relevant network statistics when both interfaces on the multi-homed host fail

inside the Linux kernel.

An interrupt can be generated if a frame arrives or there have been errors while receiving or transmitting frames. Error handling routines in the device driver code take care of these conditions gracefully. The first modification that we have performed is on this part of the device driver code. The inserted code detects specific events such as device failure or exceedingly high frame arrival rate, and informs the netdevgrp subsystem. The netdevgrp subsystem in turn responds by choosing a surrogate device and manipulates the transmit function pointer of the failed device to point to that of the surrogate device. This happens very rapidly and upper layers are not aware of this change. A transparent failover switch at the data link layer has the advantage that failure response is very quick and it minimizes packet loss regardless of whether the packets correspond to a connection-oriented or a connectionless protocol.

5.2 User Space Code

Network device drivers inside the Linux kernel export interfaces to the devices through names such as eth0 (ethernet devices) and wlan0 (wireless devices). A user space program can get/set the parameters of a device

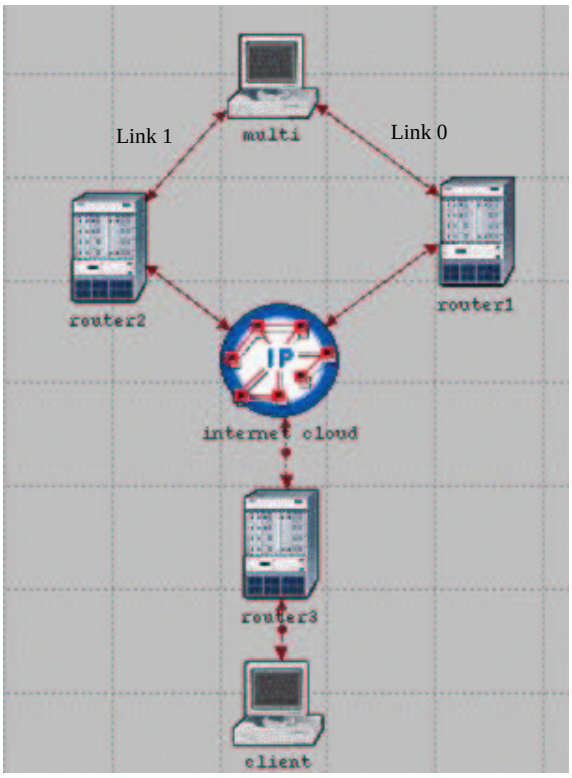


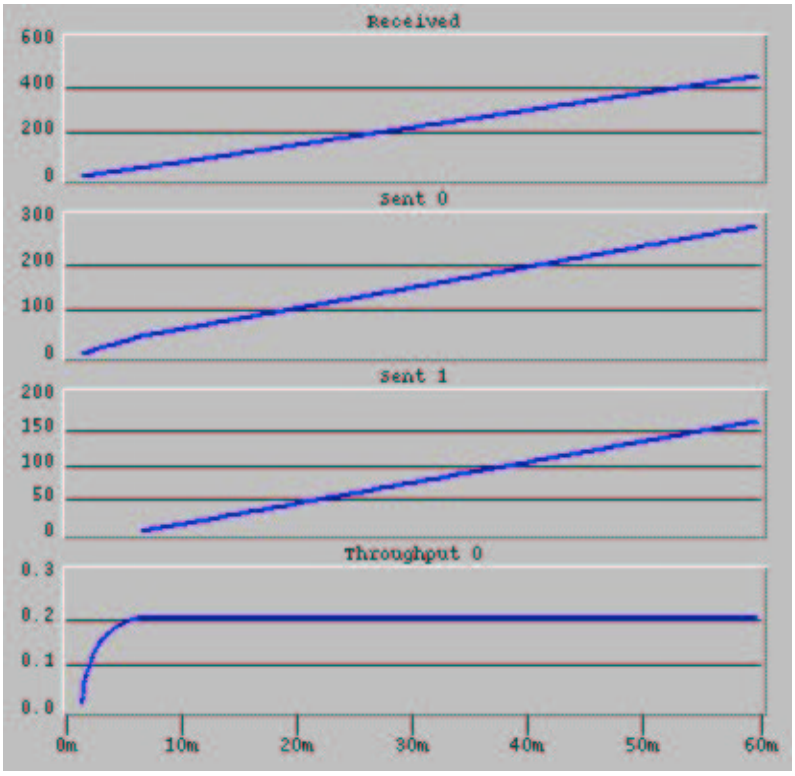
Figure 7: A network topology to model DOS attacks

by opening a TCP/IP socket and issuing ioctl commands on the socket with appropriate arguments along with the device name. This makes is easier for a user to configure the devices through programs such as *ifconfig* and *iwconfig*. We have adopted a similar design for the netdevgrp subsystem. It currently exports four devices named ndg0, ndg1, etc. In order to configure the netdevgrp subsystem, we have developed a command line tool called *ndgconfig*. A typical usage of this tool is shown in Table 1.

```
# ndgconfig create ndg0 eth0 eth1
# ndgconfig add ndg0 eth2
# ndgconfig remove ndg0 eth1
# ndgconfig destroy ndg0
```

Table 1: A typical usage of *ndgconfig*

The first line creates a new network device grouping with devices eth0 and eth1 under ndg0. If a grouping already exists under ndg0, an error is returned. New devices can be added and removed from the grouping at any time. We have currently set a limit of four devices in one network device group.



Packets received by client

Packets sent on Link 0

Packets sent on Link 1

A DOS attack causing a throughput close to maximum capacity

Figure 8: Packet statistics corresponding to a DOS attack in progress

5.3 Scope of Demonstration

In order to demonstrate the actual implementation, we have setup a host running the patched kernel with the netdevgrp support in a lab environment. Two network devices are installed in the host, each connected to a different router. Another common router delivers packets to a destination host. We have added code to inject failures in the interface cards and report these events to the netdevgrp subsystem. The results we have observed reflect those seen during simulations. However, note that this demonstration has been performed in a closed environment and the hosts are separated by small distances. We are in the process of running similar experiments over networks of a larger scale and verifying our preliminary results. It is quite possible that there may be deviations from currently observed results when experiments are conducted at that scale.

6 Conclusion and Future Work

The basic concept of network device grouping and failover support is not new. However, known techniques have implemented this in a simple manner without taking into account the network topologies and reacha-

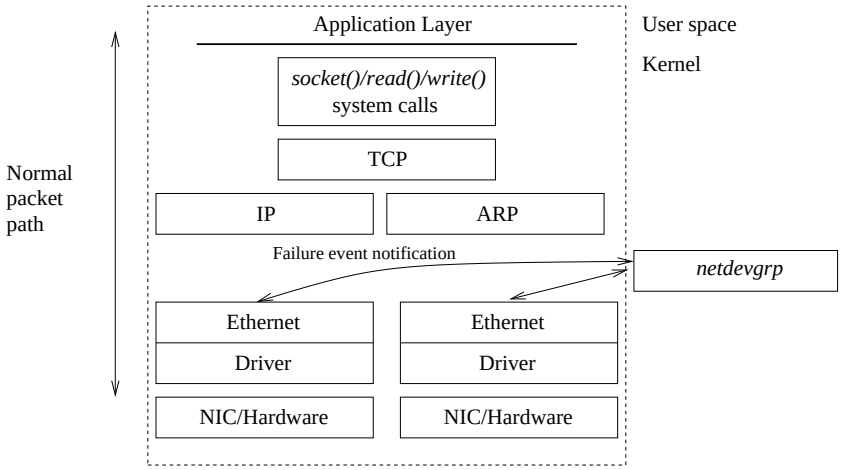


Figure 9: A logical view of the network device grouping implementation inside the Linux kernel 2.4.19

bility. This only achieves limited reliability in view of device failures. In this paper, we describe a technique that identifies some important network level criteria to perform device grouping in order to give an idea about the effectives of the grouping in a subjective sense. Appropriate device groups can achieve reliability both in terms of device failures and DOS attacks. Our technique works with device groups that may be heterogeneous such as a combination of wired and wireless network hardware. We have simulated the network device groups in OPNET to verify our hypothesis and study the feasibility. Consequently, we have implemented network device grouping as a separate subsystem in Linux and showed its effectiveness in a lab setting.

Investigation of the issue of finding a common point revealed a limitation. Some routers implement security mechanisms such as address filtering to verify that packets that are being delivered on a particular line belong to a proper address range. Such security features hinders the proper functioning of our approach. Therefore, the application of this technique is primarily in organization networks, where network reliability is an important issue.

In this paper, we have laid the basic groundwork to systematically analyze and improve network reliability. We continue to build upon this work and have identified several areas of improvement that will affect the long-term quality of this work. Some immediate research issues are as follows:

- Can the process of finding a good network device grouping be automated?

  Currently, the process of finding a good device grouping is done manually based on network maps and topological information. It may be possible to automate this process or at least part of it. This

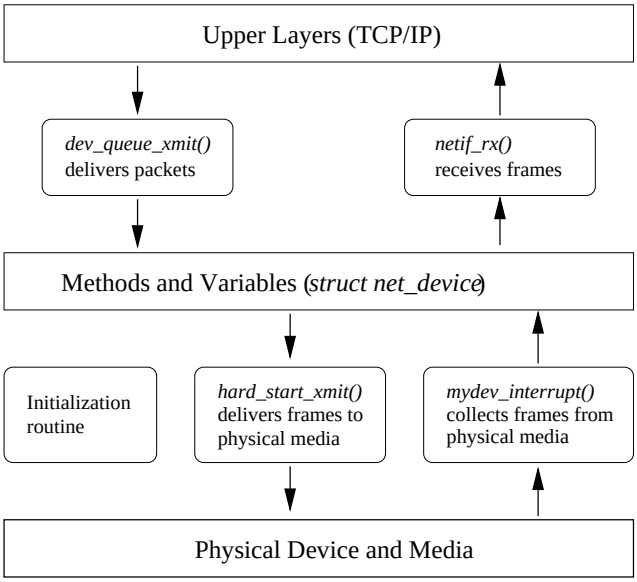


Figure 10: Basic structure of a device driver and low-level routines

would be of immense help to sysadmins and also for verification in case the groups were still selected manually.

- Can better decisions be made based on load detection?

  The choice of a surrogate device is presently a process of random selection. A preference ordering based on device speed may appear to be an improvement. However, a true measure is actually the device's current load and queue availability.

References

[1] Establishing database failover support with HACMP. IBM Websphere Documentation. http://www-3.ibm.com/software/webservers/appserv/doc/v35/ae/infocenter/was/06061410.html.

[2] Fault Tolerance of the NetEnforcer. http://www.allot.com/html/solutions_notes_tolerance.shtm.

[3] Intel's Adapter Fault Tolerance Technology. http://www.intel.com/network/connectivity/resources/technologies/fault_tolerance.htm.

[4] NetFORCE 4200C Enterprise-Class Information Management Solutions. http://www.procom.com/products/network/nf4200/Default.asp.

[5] OPNET Modeler. http://www.opnet.com/products/modeler/home.html.

[6] Stream Control Transmission Protocol (SCTP). http://www.ietf.org/rfc/rfc2960.txt.

[7] Failover for MDI devices, 1999. http://ou800doc.caldera.com/HDK_concepts/ddT_failover.html.

[8] M. K. Aguilera, W. Chen, and S. Toueg. Using the heartbeat failure detector for quiescent reliable communication and consensus in partionable networks. *TCS: Thoeretical Computer Science*, 220(1):3–30, June 1999.

[9] A. Cox. Network Buffers and Memory Management. Linux Journal, Issue 30, 1996. http://www.linuxjournal.com/article.php?sid=1312.

[10] L. Garber. Denial-of-Service Attacks Rip the Internet. *Computer*, 33(4):12–17, April 2000.

[11] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. Technical Report, ACIRI and AT&T Labs Research, February 2001.

[12] J. Xu and W. Lee. Sustaining Availability of Web Services under Distributed Denial of Service Attacks. *IEEE Transactions on Computers*, 52(2):195–208, February 2003.