A Parallel Packet Screen for High Speed Networks

Carsten Benecke
DFN-FWL*
Universität Hamburg
Vogt-Kölln-Str. 30
22527 Hamburg
benecke@fwl.dfn.de

Abstract

This paper demonstrates why security issues related to the continually increasing bandwidth of High Speed Networks (HSN) cannot be addressed with conventional firewall mechanisms. A single packet screen running on a fast computer is not capable of filtering all packets traversing a Fast/Gigabit Ethernet. This problem can be addressed by using parallel processing methods to implement a fast, scalable packet screen for Ethernets. The paper shows how hardware may be utilized to distribute the network load among such parallel packet screens. Empirical results using 'off–the–shelf' equipment indicate that this approach is usable.

1. Introduction

Firewalls are a widely employed mechanism for protecting networks from unauthorized access. They implement access control and audit functions at the interface between two or more networks with different security levels, thus enforcing a security policy.

Packet screens are simple building blocks for firewalls which allow packet based access control by checking the packet headers against filter rules. Access to hosts, networks, or services is usually controlled using a set of rules based on IP addresses, UDP/TCP ports, TCP flags, and network interfaces. Rules often include a component indicating which action is to be taken, should a packet be in contravention of the aforementioned rules. Based on this check, a packet may be either discarded, forwarded, or redirected [1].

The 'typical' topology of two networks connected via a packet screen is investigated in section 2. Performance measurements of typical activity show that a single workstation is not able to process the full traffic bandwidth in a high speed network (HSN), such as Fast Ethernet or ATM. The packet filter is therefore more than likely to represent a bottleneck to traffic in fast scaling networks with increasing bandwidth.

Section 3 presents parallel processing as a way to improve the overall throughput of a packet screen. Network equipment (e.g., hubs and switches) is used to distribute the load over the parallel devices. A simple algorithm allowing the packet filtering load to be shared among a number of parallel devices is introduced in section 4. Measurements of the prototype implementation show a significant increase in filtering performance. Some performance results are discussed in section 5. Moreover, the prototype can be adapted to full duplex Fast Ethernet networks (see section 6). Section 7 provides information regarding the reliability of the parallel packet screen.

Section 8 concludes with a discussion of open issues and ongoing improvements.

2. Firewall Topology

Figure 1 shows a simple setup of two IP networks separated by a single packet screen. All packets sent from one network to the other must pass through the packet screen. This applies the preconfigured filter rules when processing the packets. All filter rules are usually applied to the packets sequentially (in exactly the same order as they were configured). The load on the packet screen therefore not only depends on the number of packets arriving, but also on the number and order of the configured filter rules associated with each network interface.

In a high speed environment (Fast Ethernet, Gigabit Ethernet, or ATM) the single packet screen becomes a bottleneck for the communication between the individual separate networks [4]. Whereas, with typical packet sizes of 200–500 bytes, the throughput of the packet screen represents

*This work was funded by the DFN-Verein (Association for the promotion of a German Research Network) and Deutsche Telekom under project number: DT10.

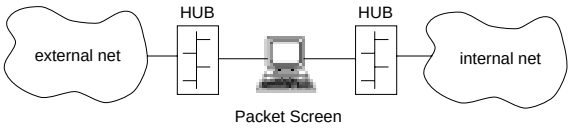

Figure 1. Packet Screen

the limiting factor, for large packet sizes a single packet screen may operate at wire speed. Applications often do not fully utilize the facilities available, 'preferring' to send smaller packets than the underlying transport mechanism can convey. Figure 2 shows the performance breakdown at the choke point of a single packet screen. The packet screen connects two 100 Mbit/s Fast Ethernet networks.

The number of packets sent and received is shown for a unidirectional UDP stream. The plots summarize the packet throughput of a direct connection through a single packet screen. The freely available tool `ip-filter`[1], modified to remove its caching mechanism to remember the action (drop or forward) associated with the previously filtered packet, provides the filter functionality. In disabling the caching, the worst case scenario is simulated: arbitrary network traffic where no two consecutive packets have equal attributes which would allow a cached decision to be of use.

The source and destination (Sun Ultra II workstations) are able to send and receive (`snd = rcv H2H`) a maximum of 34676 packets (message size of 16 byte) per second if there were no packet screens between them while the packet screen (a Sun Ultra I) limits the throughput to 12635 packets (`rcv, 0 FR`), i.e. 36%. This situation is exacerbated by increasing the number of filter rules to be applied to each packet. A humble ten filter rules (FR) reduce the throughput to 31% (`rcv, 10 FR`), a much more realistic hundred filter rules (`rcv, 100 FR`) reduces the total throughput to 3333 packets/s. This is less than 10% of the original capacity available to the source and destination hosts when communicating directly. In real environments, a caching mechanism (see above) will, to some degree, compensate for the reduced performance due to a large number of filter rules. It is therefore important to increase the filter system's underlying capacity to forward (i.e. process) packets in order to compensate for the bottlenecks described.

3. Parallel Packet Processing

The prementioned discussion shows there is a need for fast packet screens tailored to HSN. Employing faster processors for packet screening does not scale satisfactorily. Every 'single universal-processor' based approach will, eventually, become a bottleneck for a fast scaling high speed

[1] available at `http://coombs.anu.edu.au/ipfilter/`

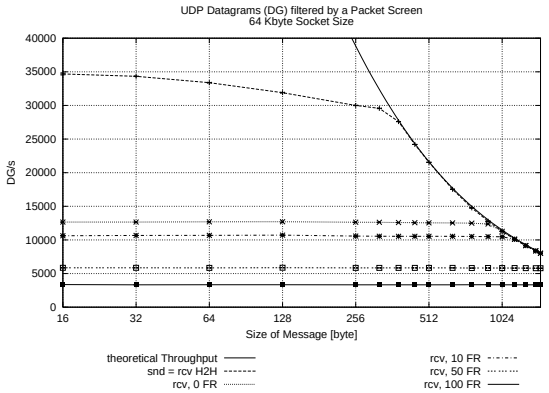

Figure 2. Performance of a Packet Screen

network. On the other hand it is doubtful if a special hardware solution (e.g., a 'Packet Screen ASIC', [9]) is flexible enough to meet future security requirements[2]. This paper recommends a much more flexible and scalable approach to increasing the total filtered packet throughput: parallel processing. This section shows how 'off–the–shelf' components, such as the workstations used for the tests described, may successfully be employed in parallel packet processing (see section 3.1).

Two important problems are addressed. First, the packet stream must be distributed among the parallel packet screens. This process must be very efficient in order to avoid creating a new bottleneck. Second, the problem of how to distribute the load among the processors involved (workstations) constituting the parallel packet screen is to be solved. Section 3.3 discusses several alternatives of an algorithm which provides this functionality.

3.1. Packet Parallelism

There are many methods of distributing the load among a number of processors[3] [8]. Two basic approaches can be taken, in order to speed up processing. First, the protocol stack of the workstation can be parallelized [10] and, second, incoming packets can be processed simultaneously [5]. The latter approach is often called as *packet parallelism* and fits well to the construction of the parallel packet screens described herein:

- All packets can be processed independently because the sequence of arriving packets needs not be pre-

[2] The adaption to new security flaws is very expensive if special purpose hardware is employed. On the other hand, software filters are easy to upgrade.

[3] The term *processor* and *workstation* are used interchangeably. As single processor workstations are employed in the tests, parallel processing requires at least two processors/workstations.

served, as IP may deliver them in any order. Furthermore, there is no need to store information pertaining to the packet's connection context unless 'dynamic filter rule' or 'context sensitive packet screen' technologies are required. In this case the inter-packet dependencies are likely to impose constraints on the process of packet distribution among the processors. Should dynamic filter rules be employed the processors may be required to exchange status information.

- Only small changes to the code of existing packet screen software are necessary. As all workstations make use of the same code[4], the only additional function needed is an algorithm to balance the packet load among the parallel workstations (see section 3.3).

- A very fine granularity of activity can be achieved, simply because each component in the parallel packet screen can process packets from any connection. Even the throughput of one single TCP-connection can be increased, since packets can be distributed among the processors.

3.2. Packet Distribution

The system shown in figure 1 can be extended to include more than one workstation for screening. An example utilizing three workstations which process arriving packets in parallel is shown in figure 3. The Ethernet hubs shown are required to propagate incoming packets to the workstations. Hubs (multi-port repeaters) usually forward all incoming packets from all interfaces (ports) to all other interfaces. This means that distribution (forwarding) of the incoming packets to the workstations is completely transparent. Since all packets are forwarded to every port on the hub, there is no need to explicitly address the packet screen components. This is a useful feature, as it means that the packet screen can, to a large extent, be hidden (if so desired). One advantage of this feature is that an 'invisible' packet screen is more difficult to attack.

3.3. Distributed Packet Selection

The employment of hubs as shown allows the packets to be distributed to all attached packet screen components. In order to increase the packet throughput, the packet load should be shared by all processors. To this end, the packets to be processed are selected by each processor, all others are discarded.

The packet selection algorithm[5] must fulfill a number of requirements:

[4]This is usually a kind of filter software which is compiled into the kernel or loaded at run time in the form of a 'loadable module'.

[5]The selection algorithm is not responsible for filtering/screening.

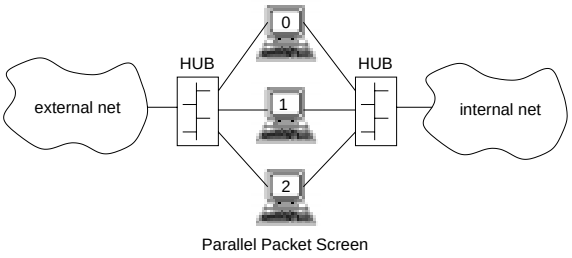

Figure 3. Parallel Packet Screen

1. The algorithm should not be centralized. Should a single entity be responsible for selecting and forwarding the packets, it may become a bottleneck for total system throughput. A distributed algorithm can, on the other hand, increase total system reliability, as no single point exists which can lower system performance (or indeed bring it to a standstill).

2. The algorithm should be independent of the number of processors. It should allow the system to be scaled so that the number of processors may be increased in order to adapt the packet screen to faster networks or more complex screening operations.

3. The algorithm should be fast. The additional overhead due to packet selection should be minimized: the selection time (T_d) should be very short in comparison with the screening time (T_t):
$\bigvee$ packets : $T_d \ll T_t$

4. The algorithm should map each incoming packet to exactly one processor, each packet will therefore be processed only once. This behavior is required, firstly for efficiency, and, secondly, in order not to forward duplicate packets or to drop packets in the selection process.

5. The algorithm should evenly distribute the packets among the processors. All processors experience the same level of utilization.

Hash-Function for Packet Selection
A hash-function usually provides a fast scalable technique to map input symbols onto a chosen range of values. A very simple, fast hash function (requirement 3) can therefore be used to map the incoming IP packets onto their respective processors. In order to select a single processor for each packet (requirement 4), the hash function must map input symbols onto a range of processor identifications (see below).

The input symbols of the hash function must be present in the packets arriving for selection. Each packet screen can

thus apply the algorithm to the packets in order to match requirement (1). The packets contain a variety of information which can be used for the hash calculation:

IP address The major drawback of using the datagram source and/or destination address in the hash calculation is that all packets with the same address (either sender or recipient) are mapped onto the same workstation / processor. Should there be only one high bandwidth connection, the mapping would cause a system bottleneck. On the other hand, some 'stateful packet screens' could require all datagrams from a host be processed by a single processor in order for context sensitive screening to be possible. In this case, the parallelism could be increased by including port numbers in the hash calculation.

IP identification This value is required to be unique for each unique datagram and is usually incremented as each datagram is sent by the IP host. The value is a good choice for distributing the datagrams among parallel facilities: even the packets of a single connection can be processed in parallel.

IP header checksum All IP hosts calculate a checksum for the complete IP header information (for calculation the checksum field itself is assumed to be zero).

The major drawback of all the IP header values discussed above is that the values cannot be used for non-IP-packets (e.g., ARP packets or packets of other protocol suites). Therefore the following values may also be used:

Frame checksum If information about the underlying network layer (layer two) protocol is available, values contained within its structure can be used. Should the underlying network be, for example, Fast Ethernet, the frame checksum can be employed in the hash calculation. Indeed the use of the frame checksum is applicable to all protocol suites being of practical relevance. Moreover, similar units can be selected for other frame types (e.g. the 'Frame Check Sequence' of an FDDI frame).

Combinations It is also possible to use a combination of the values discussed above. Indeed, in some cases it may even be necessary to combine IP addresses with port numbers.

As the objective of the hash process is to obtain an even distribution of packet processing among the packet screen components, increasing the number of input symbols (hash calculation values) may produce a more even distribution.

Algorithm for Packet Selection
Packet selection may be summarized as follows:

1. All parallel processors (packet screen with n processors) are enumerated from $0, \cdots, n-1$ to identify the processors.
2. Hubs propagate the incoming packets to all processors.
3. All processors perform the hashing based on a predefined set of packet information[6] in order to map each incoming packet to a number within the range of $0, \cdots, n-1$.
4. The processor whose ID matches the mapping of the hash function is responsible for screening the packet.
5. All other processors (which IDs do not match the hash result) discard the packet.

4. Prototype Implementation

This section discusses a prototype implementation based on the public domain software `ip-filter`, which is available for most UNIX systems.

A first approach is to use the IP-packet checksum modulo number of processors to select which packet screen component should process the packet. Further research may be required to find more appropriate hash functions for packet selection.

For best performance, we have implemented the packet selection algorithm within the Fast Ethernet device driver. Of course it is also possible to implement the algorithm as part of the filter functions, but changing the device driver permits to discard non-selected packets as fast as possible (e.g., before sending them upstream to the ip-filter module).

Next to IP packets, there are two other packet types in the TCP/IP protocol suite[7] which are transported in Ethernet frames. These are 'Address Resolution Protocol' (ARP) and 'Reverse Address Resolution Protocol' (RARP) packets [6]. As the mainstream IP packet selection should not be disrupted, a value which starts at the same frame offset as the IP checksum in Ethernet frames, is used to distribute ARP and RARP packets. At this offset a frame contains the third and fourth bytes of the (R)ARP packet's Ethernet sender address.

5. Performance of the Parallel Packet Screen

This section discusses both the empirical as well as the analytical results for the packet throughput and the gainable

[6] The prototype uses the checksum of the IP-packet modulo the total number of processors as a hash function (see section 4).

[7] Only the TCP/IP protocol suite is discussed: other suites require additional / other checks.

speedup due to parallel processing. All plots (figures 2 and 4) show confidence intervals for a 99% confidence level.

5.1. Empirical Results

Figure 4 shows the performance of parallel packet screens with two to four processors. Sender, receiver, and packet screen are connected to a Fast Ethernet at 100 Mbit/s. The `snd` plot shows the unidirectional load of packets dispatched from the sender in datagrams/s. The `rcv` plots show the number of packets received at the destination after passing through a (parallel) packet screen with `WS` processors (workstations). A single packet screen is able to forward about 12700 packets/s, which still suffices to reach the maximum possible packet throughput of a Fast Ethernet for packet sizes larger than 896 bytes.

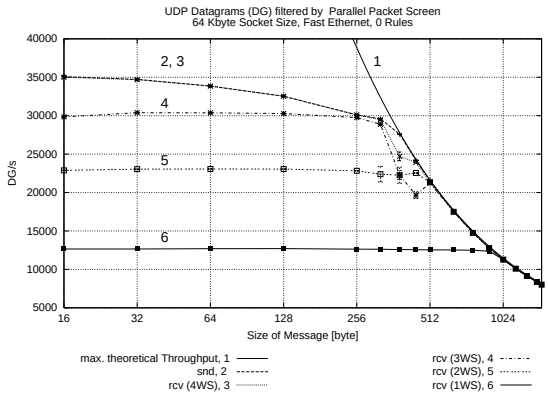

Figure 4. Performance of Parallel Packet Screen based on Fast Ethernet Hubs

A two `WS` packet screen is capable of processing about 22000 packets/s. As can be seen in figure 4, this throughput is sufficient to convey the full workload of a Fast Ethernet for packet sizes larger than 512 bytes. Three `WS` are able to forward about 30000 packets/s and four `WS` seem to be able to cope with the load offered for almost all packet sizes[8].

The small message sizes (16 to 256) almost have no influence on the throughput. The increase in packet throughput gained by parallel processing for small message sizes is listed in table 1. The value of 2.95 for a `4WS` is calculated from figure 6 since at least two simultaneous senders were required to offer an adequate load.

[8]The `snd` plot and the `rcv (4WS)` plot overlap in almost all points. The lower throughput for messages with 384 bytes is reproducible but not yet analyzed.

Processors	Speedup
1	1
2	1.82
3	2.3989
4	2.9557 (see figure 6)

Table 1: Speedup of Parallel Packet Screen

On the one hand, the increase in processing capability is not linear, as all workstations must perform selection tasks on all arriving packets. This is the limiting factor for high parallel setups (see analysis below). On the other hand, the increase is adequate so that the load presented by fast 'state of the art' workstations can be coped with.

5.2. Analytical Results

In order to predict the gainable speedup due to parallel processing it is necessary to make some assumptions. First of all, it is assumed that the algorithm discussed in section 3.3 will result in an evenly distributed packet load among all n processors. This is true for the lab setup but may not be true in some special environments. Second, all the processors are of the same kind. Once again this is true for the lab setup but in real world environments different kinds of machines may be used for parallel processing. These two assumptions result in a balanced load among all processors, e.i., the observable utilization of all processors is the same.

Making these assumptions there is a probability of $P_I = 1/n$ that any of the n parallel processors (I) select an incoming packet, and a probability of $\overline{P_I} = 1 - 1/n$ that a processor will discard an arbitrary packet. This in turn implies that for every filtered packet (service time B_{fil}) an average of n selection operations (service time B_{sel}) is necessary.

Thus the maximum packet throughput of a *single processor* ($Pmax_I$) with respect to n parallel instances is:

$$\begin{aligned} Pmax_I(n) \quad * \quad & n * B_{sel} + Pmax_I(n) * B_{fil} = U = 1 \\ Pmax_I(n) \quad = \quad & \frac{1}{nB_{sel} + B_{fil}} \end{aligned} \tag{1}$$

U denotes the utilization of the processor. The overall packet throughput of *all n processors* is therefore:

$$Pmax_n(n) = Pmax_I(n) * n = \frac{n}{nB_{sel} + B_{fil}} \tag{2}$$

Finally the gainable speedup ($G(n)$) is denoted by

$$G(n) \quad = \quad \frac{Pmax_n(n)}{Pmax_I(1)} = \frac{n(B_{sel} + B_{fil})}{nB_{sel} + B_{fil}} \tag{3}$$

$$G(n) = \frac{B_{sel} + B_{fil}}{B_{sel} + (1/n)B_{fil}}$$

with an upper limit (theoretical max. speedup) of

$$\lim_{n\to\infty} G(n) = \frac{B_{sel} + B_{fil}}{B_{sel}} \quad (4)$$

Based on the empirical results from section 5.1 B_{sel} and B_{fil} are calculated for packets with 32 bytes[9] message size (user data) and a total number of zero filter rules[10]:

$$\begin{aligned} B_{sel} &\approx 9,29420E^{-6}\frac{s}{P} \\ B_{fil} &\approx 6,96224E^{-5}\frac{s}{P} \end{aligned} \quad (5)$$

By using these values in equation 3 the theoretical speedup with respect to n is calculated in table 2:

Processors	Speedup
*1	1
2	1,7893
3	2,4281
*4	2,9557
5	3,3988
6	3,7763

Table 2: Theoretical Speedup

The values for one and four processors (marked with a *) have been used to calibrate equation 3. Thus the calculated speedup matches the empirical value of table 1. The values for two and three processors can be used to verify the analytical results. Notice that the difference between empirical and calculated speedup is less than 2% for two and three processors.

So the calculated speedup is comparable to the empirical results (see table 1). Thus equation 3 can be used to predict the gainable speedup for more than four processors. E.g., it is possible to increase the throughput to 378% by using 6 instances which is sufficient to catch up with the theoretical throughput of a Fast Ethernet (100 Mbit/s) for packet sizes larger than 200 byte of user data as there is almost no dependence on the packet size.

Equation 4 provides a value of $\approx 8,49$ as the maximum speedup for this setup. On the other hand it is quite obvious that parallel processing works even better for setups with $B_{sel} \ll B_{fil}$, e.g. by increasing the number of filter rules.

As a rule of thumb: the more filter rules the better the speedup due to parallel processing.

[9] This is an example. Other message sizes could have been used as well.

[10] packet forwarding only, no screening, unit of measurement is seconds/packet

6. Switched based Approach

The major drawback of the solution presented above, is that of the half duplex mode of the hubs employed. Hubs implement a shared segment and all connected workstations compete with each other for the available bandwidth. In conditions of heavy network load, this means that the number of collisions on the Ethernet will increase dramatically. Figure 5 shows the performance degradation originating from collisions in two networks where a sender in each network sends datagrams to a recipient in the other network via a four `WS` packet screen. The senders and recipients are all distinct machines and both datagram streams are sent simultaneously. The plots show the sums of individual throughputs for both senders and recipients. Even though the packet screen is able to filter about 37500 packets/s, many packets (medium to large sizes) are lost due to the massive collision rate at the hubs. For packet sizes of 256 to 576 there is a dramatic throughput decrease which is bigger than expected. Even though increasing the size of the packets eases the situation somewhat, the effect of collisions is still visible. The number of packets received never equals the number sent.

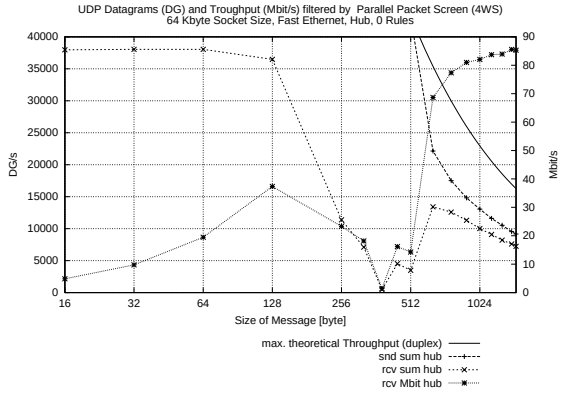

Figure 5. Bidirectional Packet Load and Effects of Collisions

In addition, most modern networks are moving from shared to switched media. If two fully switched fast Ethernet's are joined by a parallel packet screen based on hubs, the total bandwidth of 200 Mbit/s (full duplex) is reduced to at most 100 Mbit/s.

This section describes an extension to the parallel packet

screen approach allowing the use of switches to effect packet distribution. As switches support full duplex mode, the packet screen can also be driven at full duplex mode. Switches usually forward datagrams from one interface to exactly one other interface, a method is therefore needed which facilitates the forwarding of datagrams from one network to all workstations in the parallel packet screen.

The switch based approach suggested uses multicast addresses, whereby all workstations constituting the parallel packet screen are configured to receive packets for a special multicast address. Switch interfaces connected to one of the packet screen workstations are thus merged into a multicast group. The workstations are configured to reply to ARP queries for this multicast address. Furthermore, all workstations constituting the packet screen are configured with the same IP address.

A host from either network wanting to send datagrams to the other network looks up its configuration and receives the IP address of the packet screen which is the next hop address (router). The Ethernet address is received via an ARP query broadcast to any workstation attached to the switch. One of the packet screen workstations will select the packet as described above (section 3.3). Since the queried IP address is statically bound to the multicast Ethernet address, the packet screen will respond with this multicast address (see [6]). The host thus receives ('learns') the multicast address as the physical address of the next router.

Whenever the host sends datagrams containing the multicast address, the switch forwards the datagrams to all attached packet screen workstations, because they have been joined together to form a multicast group. This setup allows the use of the network (switches) to distribute the load while permitting it to be used in full duplex mode.

Figure 6 shows the performance of the switched based approach. The maximum packet throughput (about 37500 packets/s) stays the same since the filtering devices have not changed. Since no collisions occur the throughput stays almost constant until the number of packets received equals the number of packets sent (sizes ≥ 640 bytes). The maximum throughput is reached for 1472 byte messages at 189 Mbit/s.

The results demonstrate that the parallel packet screen approach is well suited to the switched network architecture.

7. Reliability

As shown it is possible to increase the packet throughput of a packet screen by parallel processing. On the other hand the failure probability also increases unless additional mechanisms are deployed. As a very simple failure model, it is assumed that a processor either works well or is subject to a "crash" failure [2], e.g., the workstation goes offline

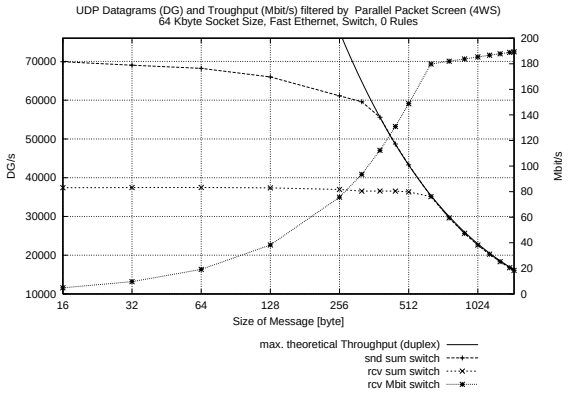

Figure 6. Bidirectional Packet Load, Switched based Approach

and is no longer able to process any packets.

This model is known as a "n of n system" where all of the n components (processors) must be alive. If one of the n processors fails, approximately every nth packet is discarded (see section 5.2). This results in a systematic packet loss.

Usually X_i indicates whether a component i of a system S is defect or intact:

$$X_i(t) = \begin{cases} 0, & \text{if component } i \text{ is intact at time } t \\ 1, & \text{if component } i \text{ is defect at time } t \end{cases} \quad (6)$$

If there is for example a probability of survival of 85% for a single processor packet screen after a certain time $t > t_0$, that is $(P(X(t) = 0) = 85\%)$, then the probability of survival of a parallel packet screen that is composed of n equal components is [7]:

$$\begin{aligned} A_{nofn}(t) &= \prod_{i=1}^{n} P(X_i(t) = 0) \quad (7) \\ A_{nofn}(t) &= (P(X_i(t) = 0))^n \end{aligned}$$

If we deploy four processors the probability of survival after the same time t decreases from 85% to $0.85^4 \approx 52\%$!

As a rule of thumb: the more parallel processors the lower the survival probability. Thus the more parallel processors the shorter the time to repair.

After a crash a systematic packet loss occurs. As about every nth packet is lost the packet throughput after a crash (P_{crash}) decreases to:

$$P_{crash}(n) = Pmax_n(n) - \frac{Pmax_n(n)}{n} = Pmax_n(n)(1 - \frac{1}{n}) \quad (8)$$

This behavior is undesirable although higher level protocols such as TCP may be able to overcome this problem. Ongoing work focuses on a more general approach.

By sending periodic "heart beat" signals the processors can monitor each other. If a crash is detected by any of the other processors, the remaining workstations can reconfigure themselves to the new number of available processors. After a short reconfiguration systematic packet losses are no longer visible. Each of the n parallel processors acts as a functional/active redundancy for all other processors. By using this algorithm it is possible to increase the availability of the parallel packet screen.

Moreover, the performance is also increased by the reconfiguration. After the reconfiguration the throughput is reduced from $Pmax_n(n)$ to $Pmax_n(n-1)$. This is still a better value than $Pcrash(n)$:

$$\begin{aligned} Pmax_n(n-1) &> Pcrash(n) \qquad (9)\\ \frac{(n-1)}{(n-1)B_{sel}+B_{fil}} &> \frac{n}{nB_{sel}+B_{fil}} - \frac{1}{nB_{sel}+B_{fil}}\\ \frac{n-1}{(n-1)B_{sel}+B_{fil}} &> \frac{n-1}{nB_{sel}+B_{fil}} \quad , n \geq 2 \end{aligned}$$

Equation 9 shows that reconfiguration is also important to increase the throughput after the crash of a processor. This result is independent of the number of processors (n). After repairing and reintegrating the crashed processor into the parallel setup another reconfiguration can restore the maximum packet throughput to $Pmax_n(n)$.

8. Conclusion

This paper discusses the need for a scalable high speed packet screen capable of filtering packets in high speed networks. Parallel processing is used to increase the throughput of the packet screen. The use of hubs and switches to distribute the load over a number of parallel filtering devices allows highly scalable parallel packet filter devices to be constructed using 'off–the–shelf' components.

Performance measurements show a significant improvement in packet processing performance, which allows secure high speed networks to be built. This technique should be combined with mechanisms to speed up proxies on bastion hosts in order to build more sophisticated high speed firewalls [4]. Moreover, ongoing work shows that the underlying algorithm (see section 3.3) is also applicable to speed up network monitors and encryption/decryption boxes for virtual private networks.

While performance can be improved using the parallel processing architecture, the method set out above does have a drawback: an associated increased failure probability. Measurements show that a fail-stop breakdown of one device significantly reduces the throughput of TCP connections. This behavior is due to the loss of data packets and acknowledgments: the distributed selection of packets is not self stabilizing. Ongoing work is directed towards group management protocols [3] for status exchange and fault detection (see section 7) with the objective of increasing both performance and availability by parallel processing.

References

[1] D. B. Chapman and E. D. Zwicky. *Building Internet Firewalls*. O'Reilly, 1995.

[2] F. Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, February 1991.

[3] F. Cristian. Reaching Agreement on Processor Group Membership in Synchronous Distributed Systems. Research Report RJ 5964 (59426), IBM Research Division, 1998.

[4] U. Ellermann and C. Benecke. Firewalls for ATM Networks. In *Proceedings: INFOSEC'COM*, Paris, June 4./5. 1998.

[5] M. Goldberg, G. Neufeld, and M. Ito. *A Parallel Approach to OSI Connection-Oriented Protocols*, pages 219–232. North-Holland, 1993.

[6] D. C. Plummer. An Ethernet Address Resolution Protocol. Request For Comments 826, Network Working Group, 1982.

[7] W. G. Schneeweiss. *Grundbegriffe für praktische Zuverlässigkeitsanalysen*. Datakontext-Verlag, Köln, 1981.

[8] C. Woodside and G. Franks. Alternative Software Architectures for Parallel Protocol Execution with Synchronous IPC. *IEEE/ACM Transactions on Networking*, 1(2):178–186, April 1993.

[9] J. Xu and M. Singhal. Design of A High-Performance ATM-Firewall. In *Proceedings: 5th ACM Conference on Computer and Communications Security*, San Francisco, California, November 4–5 1998.

[10] M. Zitterbart. *Funktionsbezogene Parallelität in transportorientierten Kommunikationsprotokollen*. Number 183 in Fortschritt-Berichte, VDI Reihe 10. VDI-Verlag, 1991.