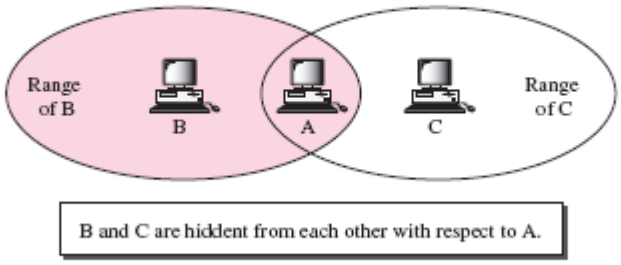
IGNOU MCA MCS-042 Solved Assignment 2011

Question 1(a) Hidden Station Problem Figure 14.10 shows an example of the hidden station problem. Station B has a transmission range shown by the left oval (sphere in space); every station in this range can hear any signal transmitted by station B. Station C has

Figure 14.10 *Hidden station problem*

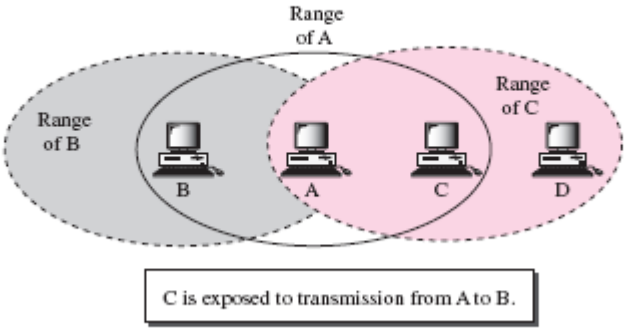


a transmission range shown by the right oval (sphere in space); every station located in this range can hear any signal transmitted by C. Station C is outside the transmission range of B; likewise, station B is outside the transmission range of C. Station A, however, is in the area covered by both B and C; it can hear any signal transmitted by B or C.

Assume that station B is sending data to station A. In the middle of this transmission, station C also has data to send to station A. However, station C is out of B's range and transmissions from B cannot reach C. Therefore C thinks the medium is free.

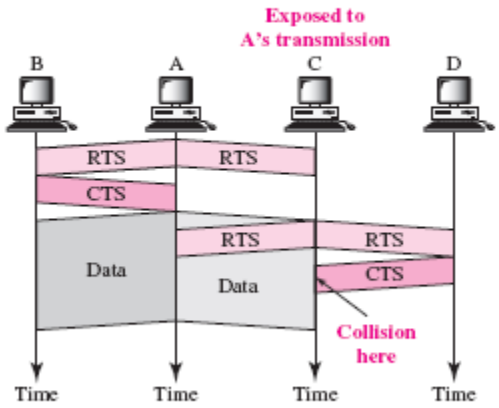
Exposed Station Problem Now consider a situation that is the inverse of the previous one: the exposed station problem. In this problem a station refrains from using a channel when it is, in fact, available. In Figure 14.12, station A is transmitting to station B. Station C has some data to send to station D, which can be sent without interfering with the transmission from A to B. However, station C is exposed to transmission from A; it hears what A is sending and thus refrains from sending. In other words, C is too onservative and wastes the capacity of the channel.

Figure 14.12 *Exposed station problem*



The handshaking messages RTS and CTS cannot help in this case, despite what you might think. Station C hears the RTS from A, but does not hear the CTS from B. Station C, after hearing the RTS from A, can wait for a time so that the CTS from B reaches A; it then sends an RTS to D to show that it needs to communicate with D. Both stations B and A may hear this RTS, but station A is in the sending state, not the receiving state. Station B, however, responds with a CTS. The problem is here. If station A has started sending its data, station C cannot hear the CTS from station D because of the collision; it cannot send its data to D. It remains exposed until A finishes sending its data as Figure 14.13 shows.

Figure 14.13 *Use of handshaking in exposed station problem*



Question 1(b)

The ALOHA Protocol

In the seventies, the ALOHA system was proposed by Norman Abramson as an effective solution to provide for wireless access to computer systems. The ALOHA-net at the University of Hawaii

employed fixed transmitters at islands located at ranges of several tens of kilometers. The main advantage of the ALOHA random access scheme was simplicity. Terminals can transmit their data regardless of the activity of other terminals. If a message is successful the base station sends an acknowledgement over a feedback channel. If the terminal does not received an acknowledgement, the terminal retransmits the message after waiting a random time. The delay is mainly determined by the probability that a packet is not received (because of interference from another transmission, called a "collision") and the average value of the random waiting time before a retransmission is made.

Collision Resolution

Later studies revealed that, for an infinite population of users and under certain channel conditions, the ALOHA system is unstable. Packets lost in a collision are retransmitted, but the retransmission again experiences a collision. This may set off an avalanche of retransmission attempts. Almost surely, the "backlog", i.e., the number of previously unsuccessful packets that need to be retransmitted, grows beyond any finite bound. One method to mitigate instability is to dynamically adapt the random waiting times of all terminals if the base station notices that many collisions occur. Examples of methods to ensure stability are Dynamic Frame Length (DFL) ALOHA, by Frits Schoute, or the Stack Algorithm by Boris Tsybakov et al. DFL uses a centralized control, mastered by the base station, while the stack algorithm is a decentralized method.

ALOHA in Mobile Radio Nets

The ALOHA concept is very commonly used in modern wireless communication systems. The call set-up procedure of almost any (analog or digital) cellular telephone system uses some kind of ALOHA random access. But the performance differs from what one would expect in a wireline network.

In a radio channel, packets may be lost because of signal fading even if no contending other signal is present. On the other hand, packets may be received successfully despite interference from competing terminals. This is called `receiver capture'. This effect has a significant influence on the throughput.

Optimum frequency reuse for ALOHA Random Access networks differs from frequency reuse for telephony, because the performance criteria differ (throughput / delay versus outage probability, respectively). The best reuse pattern for an ALOHA system is to use the same frequency in all cells.

The ALOHA Principle

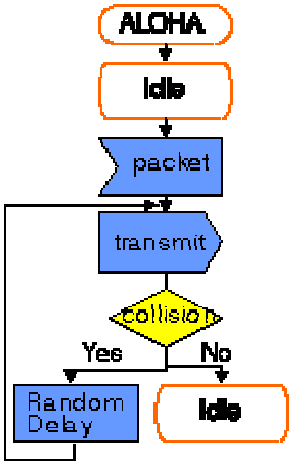


Figure: Description of terminal behavior in ALOHA random access network

In unslotted ALOHA, a transmission may start at any time. In slotted ALOHA the time axis is divided to slots. All terminals are assumed to know the times at which a new slot begins. Packets may only be transmitted at the beginning of a new slot. Slotted ALOHA has significantly better throughput than unslotted ALOHA.

Example: GSM call set-up

If the transmitter-to-receiver propagation time is large and unknown, the slot time must be equal to the packet length plus a sufficiently large guard time. In the call set-up of GSM, random access packets are substantially shorter than normal telephone speech blocks: During call set-up the propagation time is still unknown because the subscriber can be anywhere in the cell. During the call, the propagation times are measured and the terminal transmitter will compensate for it, by sending all blocks a bit in advance. A closed-loop control circuit, sending adaptive timing advance/delay feedback information, is used to ensure that the timing remains correct even if the subscribers moves in the cell.

Throughput of ALOHA Networks

To express the throughput of the ALOHA random access scheme, it is often assumed that message transmission attempts occur according to a Poisson process with rate G attempts per slot. For channels in which a transmission is successful if and and only if in that slot only a single packet transmission is present, the throughput of successful messages is equal to

- The probability of having just one message: $S = G \exp\{-G\}$,
  or equivalently,
- The attempted traffic G multiplied by the probability $\exp\{-G\}$ that no interfering message is present

Both arguments yield the well-known result for the throughput of slotted ALOHA:

$$S = G \exp\{-G\}$$

For unslotted ALOHA without capture, a test packet is destroyed by any overlapping transmission starting in the time window that

- starts one packet time before the transmission of the test packet and
- closes at the end of the transmission of the test packet.

Hence, packets transmitted over an unslotted ALOHA channel see on average twice as many interfering packets as in slotted ALOHA. In fact

$$S = G \exp\{-2G\}$$

Both unslotted and slotted ALOHA exhibit the typical behaviour that

- at low traffic (small *G*), *S* is approximately equal to *G*
- at high traffic loads (large *G*), *S* decreases to zero. Almost all packets are lost in collisions.
- one throughput value *S* corresponds to two values of *G*. The curves misleading suggest that one *G* would be stable while the other is unstable. For systems without capture it turns out, however, that the ALOHA system with a fixed retransmission procedure, independent of the history of the network, is always unstable.

ALOHA in Mobile Radio Nets

In a radio channel, packets may be received successfully despite interference from competing terminals. This is called `receiver capture'. The larger the differences in received signal power, the more likely it is that one signal is sufficiently strong to capture the receiver

The throughput becomes *G* times the probability that a particular (a priori chosen) packet is sufficiently stronger than the sum of all interfering packets.

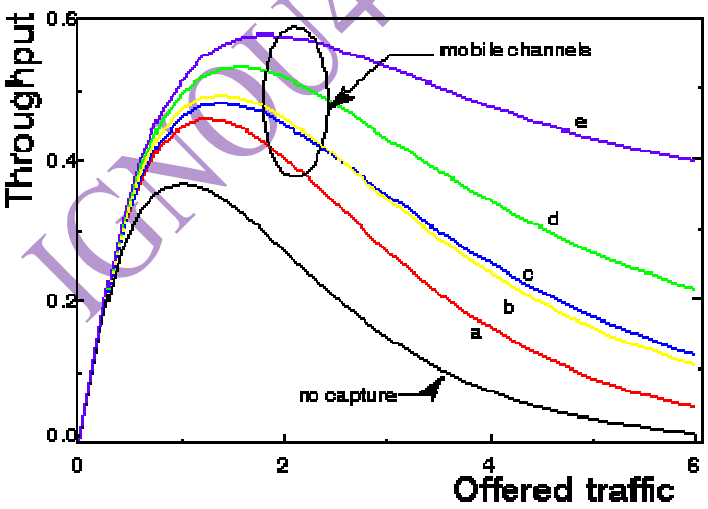
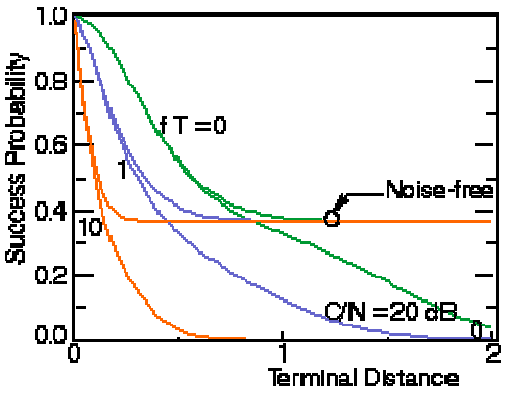


Figure: Throughput *S* of Slotted ALOHA network (in packet per time slot) versus the attempted traffic *G*.

- Without capture; any collision destroys all packets involved
- With Capture (receiver threshold is 4)
  - a: Rayleigh fading only
  - c: Shadowing and Rayleigh fading
  - e: Near far effect and Rayleigh fading, Uniform distribution of terminal is a circular cell around the receiver.
  - b,d: Some other capture models

Note that some capture models do not predict that *S* reduces to zero for large *G*.



Slotted ALOHA

Slotted ALOHA protocol. Boxes indicate frames. Shaded boxes indicate frames which are in the same slots.

An improvement to the original ALOHA protocol was "Slotted ALOHA", which introduced discrete timeslots and increased the maximum throughput. A station can send only at the beginning of a timeslot, and thus collisions are reduced. In this case, we only need to worry about the transmission-attempts within 1 frame-time and not 2 consecutive frame-times, since collisions can only occur during each timeslot. Thus, the probability of there being zero transmission-attempts in a single timeslot is:

$$Prob_{slotted} = e^{-G}$$

the probability of k packets is:

$$Prob_{slotted}k = e^{-G}(1 - e^{-G})^{k-1}$$

The throughput is:

$$S_{slotted} = Ge^{-G}$$

The maximum throughput is *1/e* frames per frame-time (reached when $G = 1$), which is approximately 0.368 frames per frame-time, or 36.8%.

Slotted ALOHA is used in low-data-rate tactical satellite communications networks by military forces, in subscriber-based satellite communications networks, mobile telephony call setup, and in the contactless RFID technologies

Question 2(a) The **leaky bucket** is an algorithm used in packet switched computer networks and telecommunications networks to check that data transmissions conform to defined limits on bandwidth and burstiness (a measure of the unevenness or variations in the traffic flow). The leaky bucket algorithm is also used in leaky bucket counters, e.g. to detect when the average or peak rate of random or stochastic events or stochastic processes exceed defined limits.

The Leaky Bucket Algorithm is based on an analogy of a bucket (figure 1) that has a hole in the bottom through which any water it contains will leak away at a constant rate, until or unless it is empty. Water can be added intermittently, i.e. in bursts, but if too much is added at once, or it is added at too high an average rate, the water will exceed the capacity of the bucket, which will overflow.

the analogue of the bucket is a counter or variable, separate from the flow of traffic, and is used only to check that traffic conforms to the limits, i.e. the analogue of the water is brought to the bucket by the traffic and added to it so that the level of water in the bucket indicates conformance to the rate and burstiness limits. This version is referred to here as the leaky bucket as a meter. In the second version [2], the traffic passes through a queue that is the analogue of the bucket, i.e. the traffic is the analogue of the water passing through the bucket. This version is referred to here as the leaky bucket as a queue. The leaky bucket as a meter is equivalent to (a mirror image of) the token bucket algorithm, and given the same parameters will see the same traffic as conforming or nonconforming. The leaky bucket as a queue can be seen as a special case of the leaky bucket as a meter

The Leaky Bucket Algorithm as a Meter

Jonathan S. Turner is credited with the original description of the leaky bucket algorithm and describes it as follows: “A counter associated with each user transmitting on a connection is incremented whenever the user sends a packet and is decremented periodically. If the counter exceeds a threshold upon being incremented, the network discards the packet. The user specifies the rate at which the counter is decremented (this determines the average bandwidth) and the value of the threshold (a measure of burstiness)” [1]. The bucket (analogous to the counter) is in this case used as a meter to test the conformance of packets [note 1], rather than as a queue to directly control them.

Another version of what is essentially the same meter version of the algorithm, the Generic Cell Rate Algorithm is described by the ITU-T in recommendation I.371 [4] and in the ATM Forum’s UNI Specification [3]. The description, in which the term *cell* is equivalent to *packet* in Turner's description [1], is given by the ITU-T as follows: “The

continuous-state leaky bucket can be viewed as a finite capacity bucket whose real-valued content drains out at a continuous rate of 1 unit of content per time unit and whose content is increased by the increment *T* for each conforming cell... If at a cell arrival the content of the bucket is less than or equal to the limit value τ, then the cell is conforming; otherwise, the cell is non-conforming. The capacity of the bucket (the upper bound of the counter) is $(T + \tau)$" [4].

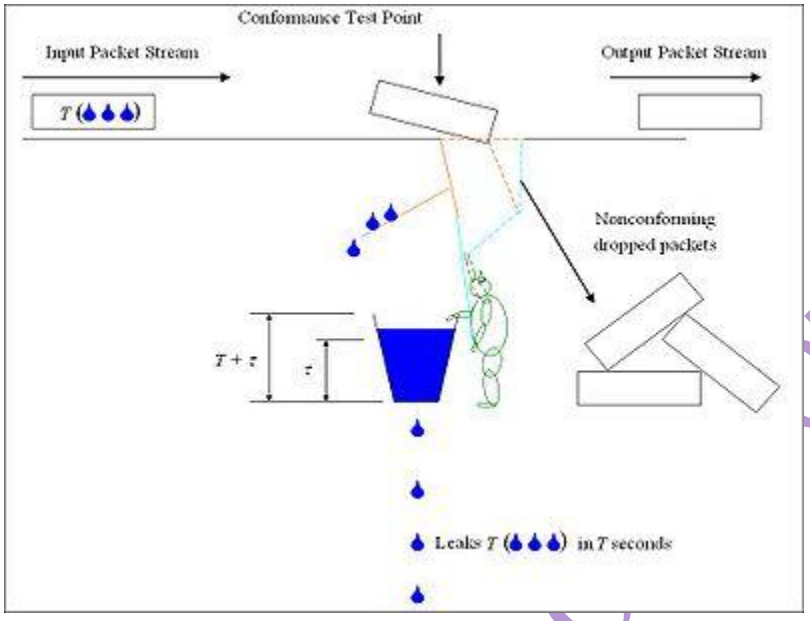


Figure 2: Traffic policing with a leaky bucket as a meter

David E. McDyson and Darrel L. Spohn provide a commentary on the description given by the ITU-T/ATM Forum. In this they state "In the leaky bucket analogy, the [ATM] cells do not actually flow through the bucket; only the check for conforming admission does" [5]. However, uncommonly in the descriptions in the literature, McDyson and Spohn also refer to the leaky bucket algorithm as a queue, going on "Note that one implementation of traffic shaping is to actually have the cells flow through the bucket"

In describing the operation of the ITU-T's version of the algorithm, McDyson and Spohn invoke a "notion commonly employed in queueing theory of a fictional 'gremlin'" [5]. This gremlin inspects the level in the bucket and takes action if the level is above the limit value τ: in policing (figure 2), it pulls open a trap door, which causes the packet to be dropped and stops its water from entering the bucket; in shaping (figure 3), it pushes up a flap, which delays the packet and prevents it from delivering its water, until the water level in the bucket falls below τ.

The difference between the descriptions given by Turner and the ITU-T/ATM Forum is that Turner's is specific to traffic policing, whereas the ITU-T/ATM Forum's is applicable to both traffic policing and traffic shaping. Also, Turner does not state that the contents of the counter should only be affected by conforming packets, and should only be incremented when this would not cause it to exceed the threshold, i.e. Turner does not

explicitly state that the bucket's capacity or counter's maximum value is finite. To make Turner's description clearly aligned with ITU-T, the statement "If the counter exceeds a threshold upon being incremented, the network discards the packet" would have to be changed to something like "If the counter would exceed a threshold upon being incremented, the network discards the packet and the counter is not incremented".

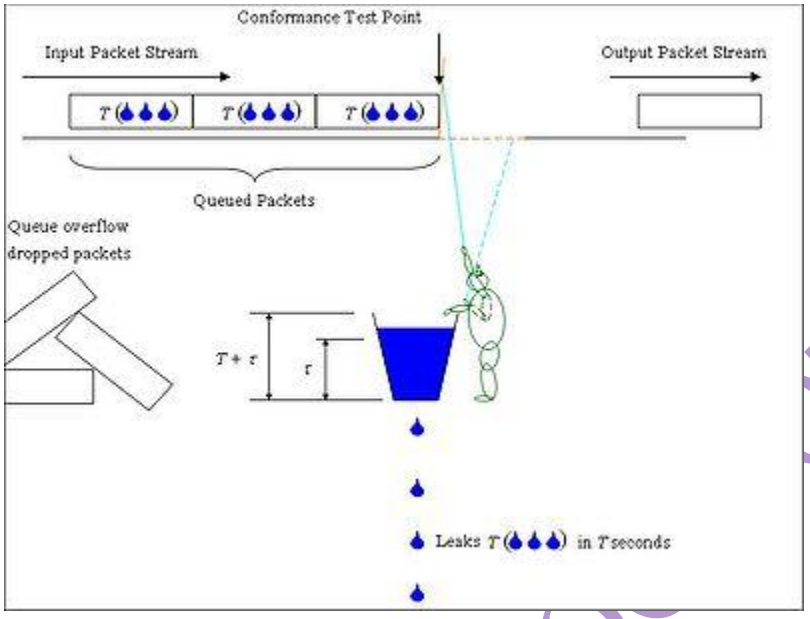


Figure 3: Traffic shaping with a leaky bucket as a meter

In fairness, the description given by Turner is in terms of a traffic policing function, where overzealousness in limiting a connection containing nonconforming packets may not be an issue. Indeed, in some contexts, such as Variable bitrate (VBR) transmissions, the loss of any one packet may corrupt the entirety of a higher layer message, e.g. an OSI Network Layer PDU. In which case, discarding all the following packets of that corrupted PDU sheds an unnecessary network load. However, it would be entirely unacceptable in traffic shaping for a packet that fails the conformance test to affect how long before conformance can next occur, i.e. if the act of testing would change how long the packet has to wait.

Neither Turner nor the ITU-T addresses the issue of variable length packets. To be fair again, the description according to the ITU-T is for ATM cells, which are fixed length packets, and Turner does not specifically exclude variable length packets. In both cases, if the amount by which the bucket content or counter is incremented for a conforming packet is proportional to the packet length, they will both account for the length and allow the algorithm to limit the bandwidth of the traffic explicitly rather than limiting the packet rate.

Concept of Operation

A description of the concept of operation of the Leaky Bucket Algorithm as a meter that can be used in either traffic policing or traffic shaping, may be stated as follows:

- A fixed capacity bucket, associated with each virtual connection or user, leaks at a fixed rate.
- If the bucket is empty, it stops leaking.
- For a packet to conform, it has to be possible to add a specific amount of water to the bucket: The specific amount added by a conforming packet can be the same for all packets, or can be proportional to the length of the packet.
- If this amount of water would cause the bucket to exceed its capacity then the packet does not conform and the water in the bucket is left unchanged.

Uses

The leaky bucket as a meter can be used in either traffic shaping or traffic policing. For example, in ATM networks, in the form of the Generic Cell Rate Algorithm, it is used to compare the bandwidth and burstiness of traffic on a Virtual Channel or Virtual Path against the specified limits. In traffic policing, nonconforming cells may be discarded (dropped) or may be reduced in priority (for downstream traffic management functions to drop if there is congestion). In traffic shaping, cells are delayed until they conform. Traffic policing and traffic shaping are commonly used in UPC/NPC to protect the network against excess or excessively bursty traffic, see bandwidth management and congestion avoidance. Traffic shaping is commonly used in the network interfaces in hosts to prevent transmissions being discarded by traffic management functions in the network.

Parameters

In the case of the leaky bucket algorithm as a meter, the limits on the traffic can be a bandwidth and a burstiness of the output The bandwidth limit and burstiness limit for the connection may be specified in a traffic contract. A bandwidth limit may be specified as a packet or frame rate, a byte or bit rate, or as an emission interval between the packets. A limit on burstiness may be specified as a jitter or delay variation tolerance, or as a maximum burst size (MBS).

Multiple sets of contract parameters can be applied concurrently to a connection using multiple instances of the leaky bucket algorithm, each of which may take a bandwidth and a burstiness limit: see Dual Leaky Bucket Controller.

Emission Interval

The rate at which the bucket leaks determines the bandwidth limit, which is referred to as the average rate by Turner [1] and the inverse of which is referred to as the emission interval by the ITU-T. It is easiest to explain what this interval is where packets have a fixed length. Hence, the first part of this description assumes this, and the implications of variable packet lengths are considered separately.

Consider a bucket that is exactly filled to the top by preceding traffic, i.e. when the maximum permitted burstiness has already occurred. The minimum interval before the next packet can conform is then the time it takes for the bucket to leak exactly the amount of water delivered by a packet, and if a packet is tested and conforms at that time, this will exactly fill the bucket once more. Thus, once the bucket is filled, the maximum rate that packets can conform is with this interval between each packet.

For variable length packets, where the amount added to the bucket is proportional to the packet length, the maximum rate at which they can conform varies according to their length: the amount that the bucket must have leaked from full for a packet to conform is the amount the packet will add, and if this is proportional to the packet length, so is the interval between it and the preceding packet that filled the bucket. Hence, it is not possible to specify a specific emission interval for variable length packets, and the bandwidth limit has to be specified explicitly, in bits or bytes per second.

Delay Variation Tolerance

It is easiest to explain what this tolerance is where packets have a fixed length. Hence, the first part of this description assumes this, and the implications of variable packet lengths are considered separately. The ITU-T define a limit value, τ that is T less than the capacity of the bucket. This limit values specifies how much earlier a packet can arrive than it would normally be expected if the packets were arriving with exactly the emission interval between them.

Imagine the following situation: A bucket leaks at 1 unit of water per second, so the limit value, τ and the amount of water added by a packet, T, are effectively in units of seconds. This bucket starts off empty, so the first packet to arrive must conform. The bucket then becomes exactly full after a number of conforming packets, N, have arrived in the minimum possible time to conform. For the last (Nth) packet to conform, the bucket must have leaked enough of the water from the preceding $N - 1$ packets ($(N - 1) * T$ seconds worth) for it to be exactly at the limit value τ at this time. Hence, the water leaked is $(N-1)T - \tau$, which because the leak is one unit per second, took exactly $(N-1)T - \tau$ seconds to leak. Thus the shortest time in which all N packets can arrive and conform is $(N-1)T - \tau$ seconds, which is exactly τ less than the time it would have taken if the packets had been arriving at exactly the emission interval.

Since the limit value τ defines how much earlier a packet can arrive than would be expected, it is the limit on the difference between the maximum and minimum delays

from the source (assuming the packets are generated with no jitter) to the point where the conformance test is being made. Hence, the use of the term Cell Delay Variation tolerance (CDVt) for this parameter in ATM.

As an example, a possible source of delay variation is where a number of connections (streams of packets) are multiplexed together at the output of a switch. Assuming that the sum of the bandwidths of these connections is less than that of the output, all of the packets that arrive can be transmitted, eventually. However, if their arrivals are independent, e.g. because they arrive at different inputs of the switch, then several may arrive at or nearly at the same time. Since the output can only transmit one packet at a time, the others must be queued in a buffer until it is their turn to be transmitted. This buffer then introduces an additional delay between a packet arriving at an input and being transmitted by the output, and this delay varies, depending on how many other packets are already queued in the buffer. A similar situation can occur at the output of a host (in the NIC) when multiple packets have the same or similar release times, and this delay can usually be modelled as a delay in a virtual output buffer.

For variable length packets, where the amount of water added by a given packet is proportional to its length, τ can't be seen as a limit on how full the bucket can be, as this varies depending on the packet size. However, the time it takes to drain from this level to empty is still how much earlier a packet can arrive than is expected, when packets are transmitted at the bandwidth limit. Thus, it is still the maximum variation in transfer delay to the point where the conformance test is being applied that can be tolerated, and thus the tolerance on maximum delay variation.

Maximum Burst Size

The limit value or delay variation tolerance also controls how many packets can arrive back-to-back at the physical layer line rate, i.e. in a burst. Hence it is possible to specify the burstiness limit as an MBS and derive the limit value τ from this or to specify it as a jitter/delay variation tolerance, and derive the MBS from this.

If the limit value is large enough, then several packets can arrive back-to-back and still conform: if the bucket starts from empty, the first packet to arrive will add T, but if, by the time the next packet arrives, the contents is below τ, this will also conform. Assuming that each packet takes δ to arrive at the line rate, then if τ (expressed as the time it takes the bucket to empty from the limit value) is equal to or greater than 'the emission interval less the minimum interarrival time, $T - \delta$, the second packet will conform even if it arrives back-to-back with the first. Similarly, if τ is equal to or greater than $(T - \delta)$ x 2, then 3 packets can arrive back-to-back, etc.

The maximum size of this burst, M, can be calculated from the emission interval, T; the maximum jitter tolerance, τ; and the time taken to transmit/receive a packet, δ, as follows

Equally, the minimum value of jitter tolerance τ that gives a specific MBS can be calculated from the MBS as follows

For variable length packets, the maximum burst size will depend on the lengths of the packets in the burst and there is no single value for the maximum burst size. However, it is possible to specify the total burst length in bytes, from the byte rate of the input stream, the equivalent byte rate of the leak, and the bucket depth.

Token Bucket Algorithm

The leaky bucket algorithm is sometimes contrasted with the token bucket algorithm. However, the above concept of operation for the leaky bucket as a meter may be directly compared with the token bucket algorithm, the description of which is given in that article as the following:

- A token is added to the bucket every 1/*r* seconds.
- The bucket can hold at the most *b* tokens. If a token arrives when the bucket is full, it is discarded.
- When a packet (network layer PDU) [*sic*] [note 1] of "n" bytes arrives, *n* tokens are removed from the bucket, and the packet is sent to the network.
- If fewer than *n* tokens are available, no tokens are removed from the bucket, and the packet is considered to be non-conformant.

This can be compared with the concept of operation, repeated from above:

- A fixed capacity bucket, associated with each virtual connection or user, leaks at a fixed rate.
- If the bucket is empty, it stops leaking.
- For a packet to conform, it has to be possible to add a specific amount of water to the bucket: The specific amount added by a conforming packet can be the same for all packets, or can be proportional to the length of the packet.
- If this amount of water would cause the bucket to exceed its capacity then the packet does not conform and the water in the bucket is left unchanged.

As can be seen, these two descriptions are essentially mirror images of one another: one adds something to the bucket on a regular basis and takes something away for conforming packets down to a limit of zero; the other takes away regularly and adds for conforming packets up to a limit of the bucket's capacity. It would also be perfectly possible to describe the process in terms of adding tokens or subtracting water for conforming packets as long as the regular process complements this, at which point it would become impossible to tell which was which: is an implementation that removes tokens regularly and adds tokens for a conforming packet an implementation of the leaky

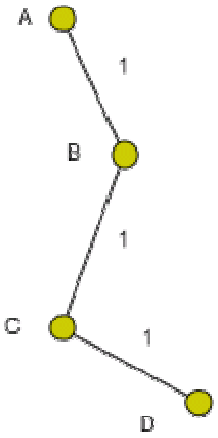
bucket or of the token bucket? In fact it is both, as these are the same basic algorithm described differently. This explains why, given equivalent parameters, the two algorithms will see exactly the same packets as conforming or nonconforming. The differences in the properties and performance of implementations of the leaky and token bucket algorithms thus result entirely form the differences in the implementations, i.e. they do not stem from differences in the underling algorithms.

The points to note are that the leaky bucket algorithm, when used as a meter, can allow a conforming output packet stream with jitter or burstiness, can be used in traffic policing as well as shaping, and can be implemented for variable length packets.

Question 2 b)

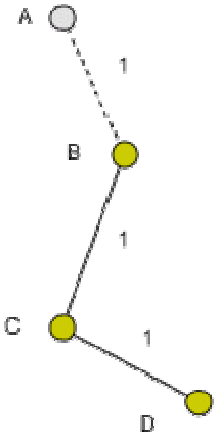
Count-To-Infinity Problem

Followed illustration shows an imagined network and denotes the distances from router A to every other router. Until now every thing works fine.



B	C	D
1	2	3

The illustration shows that link (A, B) is broken. Router B observed it, but in his routing table he sees, that router C has a route to A with 2 hops.
The problem is, that router B doesn't know that C has router B as successor in his routing table on the route to A.
That occurs followed count-to-infinity problem. B actualizes his routing table and takes the route to A over router C.
In the next picture, we can see the new distances to A. In C's routing the route to A contains router B as next hop router, so if B has increase his costs to A, C is forced to do so. Router C increases his cost to A about B + 1 = 4.
Now we see the consequence of the distributed Bellman-Ford protocol: Because router B takes the path over C to A, he reactualizes his routing table and so on!
At the end this problem is going to immobilize the whole network.



B	C	D
3	2	3

B	C	D
3	4	3

B	C	D
5	4	5

Question 2 c)

A **link-state routing protocol** is one of the two main classes of routing protocols used in packet switching networks for computer communications, the other major class being the distance-vector routing protocol. Examples of link-state routing protocols include OSPF and IS-IS.

The link-state protocol is performed by every *switching node* in the network (i.e. nodes that are prepared to forward packets; in the Internet, these are called routers). The basic concept of link-state routing is that every node constructs a *map* of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical *path* from it to every possible destination in the network. The collection of best paths will then form the node's routing table.

Routed protocols are transported by routing protocols across an internetwork. In general, routed protocols in this context also are referred to as network protocols. These network protocols perform a variety of functions required for communication between user applications in source and destination devices, and these functions can differ widely among protocol suites. Network protocols occur at the upper five layers of the OSI reference model: the network layer, the transport layer, the session layer, the presentation layer, and the application layer.

Confusion about the terms *routed protocol* and *routing protocol* is common. Routed protocols are protocols that are routed over an internetwork. Examples of such protocols are the Internet Protocol (IP), DECnet, AppleTalk, Novell NetWare, OSI, Banyan VINES, and Xerox Network System (XNS). Routing protocols, on the other hand, are protocols that implement routing algorithms. Put simply, routing protocols are used by intermediate systems to build tables used in determining path selection of routed protocols. Examples of these protocols include Interior Gateway Routing Protocol (IGRP), Enhanced Interior Gateway Routing Protocol (Enhanced IGRP), Open Shortest Path First (OSPF), Exterior Gateway Protocol (EGP), Border Gateway Protocol (BGP), Intermediate System-to-Intermediate System (IS-IS),

Question 3 i) A subnet (short for "subnetwork") is an identifiably separate part of an organization's network. Typically, a subnet may represent all the machines at one geographic location, in one building, or on the same local area network (LAN). Having

an organization's network divided into subnets allows it to be connected to the Internet with a single shared network address. Without subnets, an organization could get multiple connections to the Internet, one for each of its physically separate subnetworks, but this would require an unnecessary use of the limited number of network numbers the Internet has to assign. It would also require that Internet routing tables on gateways outside the organization would need to know about and have to manage routing that could and should be handled within an organization.

The Internet is a collection of networks whose users communicate with each other. Each communication carries the address of the source and destination networks and the particular machine within the network associated with the user or host computer at each end. This address is called the IP address (Internet Protocol address). This 32-bit IP address has two parts: one part identifies the network (with the *network number*) and the other part identifies the specific machine or host within the network (with the *host number*). An organization can use some of the bits in the machine or host part of the address to identify a specific subnet. Effectively, the IP address then contains three parts: the network number, the subnet number, and the machine number.

The standard procedure for creating and identifying subnets is provided in Internet Request for Comments 950.

The 32-bit IP address is often depicted as a dot address (also called *dotted quad notation*) - that is, four groups (or quads) of decimal numbers separated by periods. Here's an example:

```
130.5.5.25
```

Each of the decimal numbers represents a string of eight binary digits. Thus, the above IP address really is this string of 0s and 1s:

```
10000010.00000101.00000101.00011001
```

As you can see, we inserted periods between each eight-digit sequence just as we did for the decimal version of the IP address. Obviously, the decimal version of the IP address is easier to read and that's the form most commonly used.

Some portion of the IP address represents the network number or address and some portion represents the local machine address (also known as the *host number* or address). IP addresses can be one of several classes, each determining how many bits represent the network number and how many represent the host number. The most common class used by large organizations (Class B) allows 16 bits for the network number and 16 for the host number. Using the above example, here's how the IP address is divided:

```
<--Network address--><--Host address-->
          130.5     .          5.25
```

If you wanted to add subnetting to this address, then some portion (in this example, eight bits) of the host address could be used for a subnet address. Thus:

```
<--Network address--><--Subnet address--><--Host address-->
          130.5      .       5           .      25
```

To simplify this explanation, we've divided the subnet into a neat eight bits but an organization could choose some other scheme using only part of the third quad or even part of the fourth quad.

Once a packet has arrived at an organization's gateway or connection point with its unique network number, it can be routed within the organization's internal gateways using the subnet number. The router knows which bits to look at (and which not to look at) by looking at a subnet mask, which is a screen of numbers that tells you which numbers to look at underneath. In a binary mask, a "1" over a number says "Look at the number underneath"; a "0" says "Don't look." Using a mask saves the router having to handle the entire 32 bit address; it can simply look at the bits selected by the mask.

Subnets

A. A subnet is a physical segment in a TCP/IP environment that uses IP addresses derived from a single network ID.

B. Each segment uses a different network ID or subnet ID.

C. Subnetting offers several advantages.

1. Allows you to mix different technologies such as Ethernet and Token Ring
2. Allows you to overcome limitations of current technologies, such as number of hosts per segment
3. Allows you to reduce network congestion by redirecting traffic and reducing broadcasts

Implementing Subnetting

A. Use these guidelines for determining subnet requirements.

1. Determine the current and future number of physical segments on your network.
2. Determine the current and future number of host addresses required for each physical segment.
3. Based on these requirements, define one subnet mask for the entire network, a unique subnet ID for each physical segment, and a range of host IDs for each subnet.

Supernetting

A. Supernetting borrows bits from the network ID and masks them as the host ID.

B. Supernetting combines several Class C addresses which each accommodate 254 hosts to accommodate larger numbers of hosts.

C. Classless Inter-Domain Routing (CIDR) is used to collapse multiple Class C addresses into one network ID.

D. Supernetting reduces entries in routing tables.

QUESTION 4 (I) Segments A segment is a set of extents allocated for a certain logical structure. The following table describes the different types of segments.

Segment	Description
Data segment	Each nonclustered table has a data segment. All table data is stored in the extents of the data segment. For a partitioned table, each partition has a data segment. Each cluster has a data segment. The data of every table in the cluster is stored in the cluster's data segment.
Index segment	Each index has an index segment that stores all of its data. For a partitioned index, each partition has an index segment.
Temporary segment	Temporary segments are created by Oracle when a **SQL** statement needs a temporary work area to complete execution. When the statement finishes execution, the extents in the temporary segment are returned to the system for future use.
Rollback segment	If you are operating in automatic undo management mode, then the database server manages undo space using tablespaces. Oracle Corporation recommends that you use "Automatic Undo Management" management. However, if you are operating in manual undo management mode, then one or more rollback segments for a database are created by the database administrator to temporarily store undo information. The information in a rollback segment is used during database recovery: • To generate read-consistent database information • To **roll back** uncommitted **transaction**s for users

In this lesson, you will learn how two TCP devices synchronize using three way handshake (3 way handshake) and what are the three steps of a TCP three way handshake and how two TCP devices synchronize.

Before the sending device and the receiving device start the exchange of data, both devices need to be synchronized. During the TCP initialization process, the sending device and the receiving device exchange a few control packets for synchronization purposes. This exchange is known as a three-way handshake.

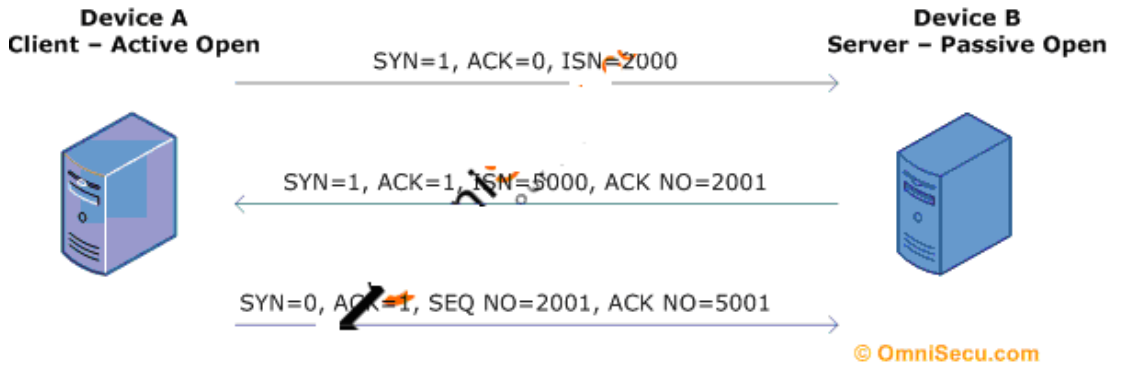
The three-way handshake begins with the initiator sending a TCP segment with the SYN control bit flag set.

TCP allows one side to establish a connection. The other side may either accept the connection or refuse it. If we consider this from application layer point of view, the side that is establishing the connection is the client and the side waiting for a connection is the server.

TCP identifies two types of OPEN calls:

Active Open. In an Active Open call a device (client process) using TCP takes the active role and initiates the connection by sending a TCP SYN message to start the connection.

Passive Open A passive OPEN can specify that the device (server process) is waiting for an active OPEN from a specific client. It does not generate any TCP message segment. The server processes listening for the clients are in Passive Open mode.



TCP Three-way Handshake

Step 1. Device A (Client) sends a TCP segment with SYN = 1, ACK = 0, ISN (Initial Sequence Number) = 2000.

The Active Open device (Device A) sends a segment with the SYN flag set to 1, ACK flag set to 0 and an Initial Sequence Number 2000 (For Example), which marks the beginning of the sequence numbers for data that device A will transmit. SYN is short for SYNchronize. SYN flag announces an attempt to open a connection. The first byte transmitted to Device B will have the sequence number ISN+1.

Step 2. Device B (Server) receives Device A's TCP segment and returns a TCP segment with SYN = 1, ACK = 1, ISN = 5000 (Device B's Initial Sequence Number), Acknowledgment Number = 2001 (2000 + 1, the next sequence number Device B expecting from Device A).

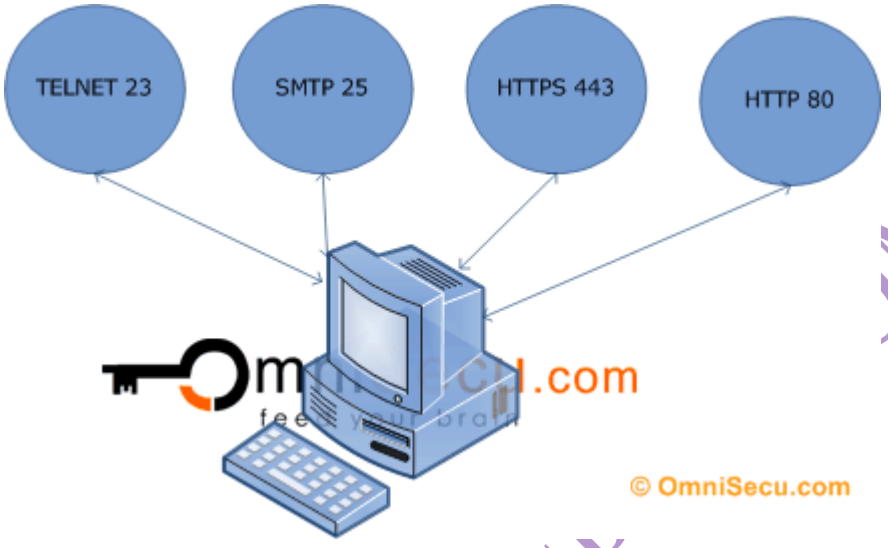
Step 3. Device A sends a TCP segment to Device B that acknowledges receipt of Device B's ISN, With flags set as SYN = 0, ACK = 1, Sequence number = 2001, Acknowledgment number = 5001 (5000 + 1, the next sequence number Device A expecting from Device B)

This handshaking technique is referred to as the Three-way handshake or SYN, SYN-ACK, ACK.

After the three-way handshake, the connection is open and the participant computers start sending data using the sequence and acknowledge numbers.

Transport Layer protocols (TCP and UDP) are responsible for supporting multiple network applications at the same instance and these applications can send and receive network data

simultaneously. Transport Layer Protocols are capable of doing this by making use of application level addressing, known as port numbers. The data from different applications operating on a network device are multiplexed at the sending device using port numbers and demultiplexed at the receiving device, again using port numbers.



The two 16 bit fields in the TCP Header**, Source port and Destination port identifies the port number which the application is listening at the sending device and receiving device. Since port number is a 16 bit number, the maximum possible value is 65535 ((2^16)-1).**

The port numbers are divided into three ranges.

The Well Known Ports are those in the range 0 - 1023. The Well Known Ports are assigned by the IANA for major protocols.

The Registered Ports are those in the range 1024 - 49151.

The Private Ports are those in the range 49152 - 65535.

Question 4 (ii) The main task of the Transmission Control Protocol is simple: packaging and sending data. Of course, almost every protocol packages and sends data. What distinguishes TCP from these protocols is the sliding window mechanism that controls the flow of data between devices. This system not only manages the basic data transfer process, it is also used to ensure that data is sent reliably, and also to manage the flow of data between devices to ensure that data is transferred efficiently without either device sending data faster than the other can receive it.To enable TCP to provide the features and quality of data transfer that applications require, the protocol had to be enhanced beyond the simplified data transfer mechanism we saw in preceding sections. Extra "smarts" needed to be given to the protocol to handle potential problems, and changes to the basic way that devices send data were implemented to avoid inefficiencies that might otherwise have resulted.

In this section I describe how TCP ensures that devices on a TCP connection communicate in a reliable and efficient manner. I begin with an explanation of the basic method by which TCP detects lost segments and retransmits them. I discuss some of the

issues associated with TCP's acknowledgment scheme and an optional feature for improving its efficiency. I then describe the system by which TCP adjusts how long it will wait before deciding that a segment is lost. I discuss how the window size can be adjusted to implement flow control, and some of the issues involved in window size management. This includes a look at the infamous “Silly Window Syndrome” problem, and special heuristics for addressing issues related to small window size that modify the basic sliding windows scheme. I conclude with a discussion of TCP's mechanisms for handling and avoiding congestion.

In a connection between a client and a server, the client tells the server the number of bytes it is willing to receive at one time from the server; this is the client's *receive window*, which becomes the server's *send window*. Likewise, the server tells the client how many bytes of data it is willing to take from the client at one time; this is the server's *receive window* and the client's *send window.*

The use of these windows is demonstrated in the topic discussing TCP's basic data transfer and acknowledgment mechanism. However, just as the example in that topic was simplified because I didn't show what happens with lost segments, there's another way that it doesn't reflect the real world conditions of an actual Internet: the send and receive window sizes never changed during the course of communication.

Impact of Buffer Management on TCP Window Size

To understand why the window size may fluctuate, we need to understand what it represents. The simplest way of considering the window size is that it indicates the size of the device's receive buffer for the particular connection. That is, window size represents how much data a device can handle from its peer at one time before it is passed to the application process. Let's consider the aforementioned example. I said that the server's window size was 360. This means the server is willing to take no more than 360 bytes at a time from the client.

When the server receives data from the client, it places it into this buffer. The server must then do two distinct things with this data:

- **Acknowledgment:** The server must send an acknowledgment back to the client to indicate that the data was received.
- **Transfer:** The server must process the data, transferring it to the destination application process.

It is critically important that we differentiate between these two activities. Unfortunately, the TCP standards don't do a great job in this regard, which makes them very difficult to understand. The key point is that in the basic sliding windows system, data is acknowledged when received, but ***not necessarily*** immediately transferred out of the buffer. This means that is possible for the buffer to fill up with received data faster than

the receiving TCP can empty it. When this occurs, the receiving device may need to adjust window size to prevent the buffer from being overloaded.

Question 5

Data encryption is used pervasively in today's connected society. The two most basic facets of modern day data encryption are data privacy and authentication. As modern society becomes more connected, and more information becomes available there is a need for safeguards which bring data integrity and data secrecy. In addition, authenticating the source of information gives the recipient, with complete certainty that the information came from the original source and that it has not been altered from its original state. Both, the needs for information privacy and data authentication has motivated cryptography.
The Xentec *X_3DES Triple-DES Cryptoprocessor* (Figure 5) supports the Spartan-II FPGA family . The features are:

- Implemented according to the X9.52 standard
- Implementation based on NIST certified DES core
- Two independent keys supported
- Both encryption and decryption supported
- Encryption and decryption latency is 48 clock cycles and throughput is 16 clock cycles
- No dead cycles for key loading or mode switching
- Fully synchronous design
- Available as a fully functional and synthesizable VHDL or Verilog soft-core
- Test benches provided
- Xilinx netlist available

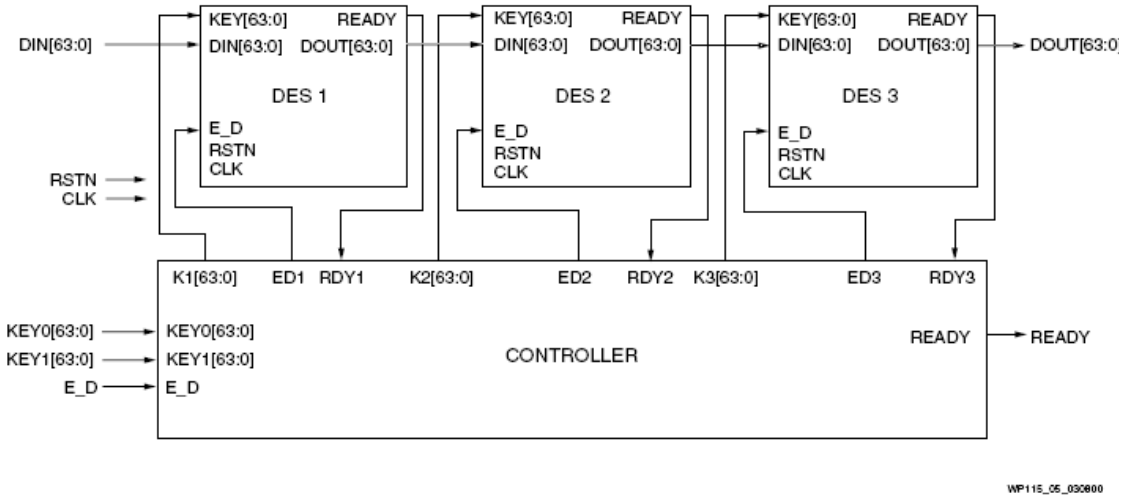


Figure 5: **The Triple-DES Implementation** *(courtesy: Xentec, Inc.)*

Programmable Logic Devices (PLDs) have always been viewed as being expensive, slower, and less feature rich than comparable ASICs. This limited their success in penetrating the ASSP market. Bringing programmability and its traditional benefits to a design solution is always an expensive proposition and PLDs have traditionally lost the battle to the costoptimized custom solution or ASIC. The use of innovative process technologies has leveled this playing field. This approach has allowed PLDs to significantly reduce die sizes, and therefore lower the cost of the overall solution. This rapid transition in process technology has allowed

PLD vendors to service the needs of today's ASSP designers. The Spartan-II FPGAs offer more than 100,000 system gates at under $10, hence making them the most cost-effective PLD solution ever offered. They address low cost and fast time-to-market, but more importantly integrate powerful new system-level features that provide an attractive solution for today's system level designer. The associated features include SelectI/O™, Block SelectRAM™, Distributed RAM, DLL (delay-locked loop) circuits, clock speeds up to 200 MHz, and aggressive power management.

The extensive features position the Spartan-II family as a low-cost, high-performance ***programmable ASSP*** alternative and expand the time-to-market advantage that PLDs traditionally offer. There are some significant advantages in using the DES/TDES soft-IP in a Spartan-II device.

Performance

FIPS 46 allows implementation of the cryptographic algorithm in software, firmware, hardware, or any combination thereof to enable more flexible, cost-effective implementations. However, a hardware implementation runs inherently faster (even by an order of magnitude) than a software implementation. The bit data rate is four times the clock rate, i.e., 100 MHz = 400 Mbps. The hardware implementation of the DES and TDES IP ported on a Spartan-II device uses 272 CLBs and provides performance upwards of 100 MHz. Most real-world consumer applications such as real-time video require only DES (and not TDES) due to performance degradation, because of extra computation.

The Xilinx Spartan-II FPGA hardware can speed the factorization of large numbers, by setting up four memory banks that are accessed simultaneously, and hence offering parallel computing. This approach increases execution speeds in repetitive calculations, required for sieving.

NIST approved

The DES soft IP for Spartan-II FPGAs is NIST approved and meets all government standards. The TDES standard is still in the process of being approved by NIST, and is a standard that all ASSPs are required to meet.

Value addition of a reconfigurable fabric

The Spartan-II family accommodates specification changes that can be easily adopted, in post volume production as part of the solution. Conflicting specifications and lack of a clear direction create the need for programmable ASSP solutions. The reprogrammable FPGA fabric permits the use of DES and TDES as needed, and allows ability to include any specification changes made by the government later. It would be nearly impossible and cost-prohibitive for an ASSP vendor to cater to all the various specifications. However, at the same time betting on the success of one single product may preclude them from being successful in the marketplace. These conditions create many opportunities for the Spartan-II family—the industry's first programmable ASSP. Because of the high profit margins involved with these products, designers can easily continue using programmable ASSPs in volume production.

The programmable fabric can also encode and decode with larger and better transformation blocks. The key can be changed within the fabric in quick intervals. Additionally if a key is ever broken, the Xilinx solution can be reconfigured instantly. Use of reconfigurable devices lets the algorithms be changed or swapped entirely, which has become a requirement in some multialgorithmic
cryptographic systems (such as Secure Socket Layer). Hence, AES hardware can be designed now, even before the standard is established.

Most stand-alone ASSPs never behave as expected, due to reasons ranging from bugs in the silicon, system integration issues, software drivers, or even user errors. Irrespective of the cause, verifying and identifying device problems can be very difficult with ASSPs, but a lot easier with programmable ASSPs. Having been built on the fabric of a proven FPGA technology and having silicon that is pre-verified and guaranteed to perform, potential problems in the Spartan-II family are narrowed down to a software-only issue. Xilinx provides powerful

==========

==========