Chapter 7: Conclusions

7.1. Introduction

Chapter 1 presented a survey of congestion control techniques and a set of desirable characteristics of any congestion control scheme. In this chapter, we review the results from the preceding chapters, and examine the extent to which we have been successful in our design effort. We also examine the weaknesses in our work, and areas for future work.

7.2. Summary of the thesis

We define congestion as the loss of utility to a network client due to an overload in the network. This motivates the design of congestion control schemes, which allow users to gain utility from the network either by reserving resources to prevent overload, or by reacting to increased network loads. While congestion control can, and should, operate at a number of time scales concurrently, we restrict the scope of this thesis to reactive control schemes that operate at time scales ranging from less that one round trip time to a few round trip times. This corresponds to the design of a scheduling discipline that operates at all queueing points, and a transport level flow control protocol that is executed at all the hosts. In the course of the thesis, we describe and analyze the Fair Queueing scheduling discipline (FQ) and the Packet-Pair flow control protocol (PP), and claim that these mechanisms provide the required functionality and performance.

Chapter 2 introduced FQ as a way to provide a fair share of switch resources to competing conversations. We defined the notion of min-max fairness, and showed how this leads naturally to the FQ algorithm. A simple analysis considered the delay distribution of a Telnet conversation competing with FTP conversations at a FQ server. Chapter 3 studied data structures and algorithms for the efficient implementation of FQ, particularly the packet buffering scheme. We concluded that, if packet losses are few, then a simple ordered linked list is the best alternative. If there can be many losses, then a per-conversation linked list data structure is best. In Chapter 4, the traffic delinking property of FQ is used to build a deterministic model for a conversation in a network, and, from a series of lemmas, we derive the paired-packet probe. This is used to design the first version of the PP protocol, which constantly adjusts the data transmission rate to measured changes in the bandwidth-delay product. In Chapter 5 we recognize the inadequacies of the deterministic model, and propose a stochastic extension. This motivates a control theoretic approach to decide how best to use the series of packet pair probes to derive a stable flow control protocol. Practical issues, such as the unavailability of noise variances, motivated the design of a fuzzy prediction scheme. The resulting protocol allows users to obtain a desired delay-throughput tradeoff by choosing a setpoint, and dynamically adjusting the packet transmission rate so that the setpoint is maintained. As in all design problems, the proof of the pudding is in the eating. We showed in Chapter 6 that PP can match, or better, the performance of some widely known flow control schemes in a variety of scenarios, some of which are specifically designed to contradict our assumptions.

We now consider the role of FQ and PP as congestion control algorithms. The FQ scheduling discipline provides a number of advantages. First, it protects well-behaved users from ill-behaved ones, so that well-behaved users can get utility from the network even in the presence of ill-behaved or malicious users (this was the main purpose of the algorithm as proposed by Nagle (101)). Second, because it provides the equivalent of per-channel queueing, users can choose their own delay-throughput tradeoff. As described in §5.3.1, by choosing a setpoint for the queue size at the bottleneck, users can trade low delay for possible loss of throughput. Such tradeoffs are not possible with a FCFS discipline. (Even the selective DECbit protocol can only enforce a global tradeoff, because the FCFS discipline does not allow it to provide different users with different queueing delays.) Third, by partially delinking the traffic characteristics of the sources, FQ allows each user to probe the network state. This allows for sophisticated flow control that can use this information to choose an appropriate sending rate. Finally, FQ switches give incentives for sources to do more sophisticated flow control. As the simulation results for scenario 5 in Chapter 6 showed, sources that change to PP from JK or generic flow control get better performance.

The PP protocol leverages off FQ to do intelligent flow control. It too provides utility to users in several ways. First, it allows users to choose a bandwidth-delay tradeoff that corresponds to their utility function. A user's utility translates to a choice of setpoint, and PP ensures that the flow control tracks the setpoint to the extent allowed by control delays. Second, PP allows FTP sources to maintain their throughput even if the conversation has a large round trip propagation delay, or there is a lot of cross traffic. As we saw in scenario 8, the other popular flow control schemes we investigated do not perform too well under such circumstances. Finally, the protocol adapts quite rapidly to changes in the network state. Thus, even short periods where bandwidth is available at the bottleneck can be utilized.

Thus, both FQ and PP are successful congestion control schemes, in the sense that they allow a user to gain as much utility as possible from the network, even when it is overloaded.

7.3. Requirements re-examined

We now re-examine the seven requirements for a congestion control scheme as presented in Chapter 1, and see, using the analysis of Chapters 2-5 and the simulations of Chapter 6, to what extent these requirements have been met.

7.3.1. Efficiency

We desire a congestion control scheme to be efficient in two ways: not to consume excessive amounts of resources, such as network bandwidth and switch CPU time, and second, to allow the maximum possible utilization of network capacity.

PP does not place any overhead on the net amount of data transferred in the network, since additional probe or state-exchange packets are not used. Concerning the switch CPU time requirement, with PP, switches can be completely passive. Unlike the DECbit scheme, they need not set bits, nor compute averages. FQ implementation looks complicated at first glance, and it seems as if it may take a lot of switch CPU cycles. However, as we showed in Chapter 3, the additional overhead is not exceedingly large. With an efficient implementation scheme, we feel that the benefits of doing FQ outweigh its costs.

With the PP and FQ schemes, the network bandwidth is not underutilized (as it is, for example, with the DECbit schemes in Scenario 6). Since FQ is work conserving, it does not keep bandwidth idle if there are packets waiting to be served. Further, PP adjusts the packet transmission rate so that the bottleneck queue is never emptyl; hence a source obtains all the throughput allocated to it as its fair share. Indeed, we note that in Chapter 6, in all the scenarios, all the available bottleneck capacity is utilized.

Thus, we meet both the efficiency criteria mentioned in Chapter 1.

7.3.2. Heterogeneity

We require that the congestion control scheme accommodate heterogeneity in packet size, transport layer protocols and type of service requirements. FQ clearly allows for heterogeneity in packet sizes: indeed, that is one reason why we modified Round-Robin to obtain FQ. Second, by allowing users to choose their own delay-throughput tradeoffs, PP with FQ can accommodate a variety of service requirements. Finally, the results of scenario 5 indicate that PP can coexist with other protocols, and perform as well as, or better, than they can. Thus, we have satisfied our heterogeneity requirements.

The only caveat to the above is that, for PP to work correctly, every *potential* bottleneck must implement FQ or a similar round-robin-like service discipline. Even if a single bottleneck serves packets using FCFS, then the values reported by the probes will no longer be valid, and PP is no longer feasible. However, if the FCFS bottlenecks enforce some sort of rate control, and can stamp packets with the current service rate, then PP can be salvaged. This restriction is a major hurdle to the implementation of PP in current, FCFS networks.

7.3.3. Ability to deal with ill-behaved sources

Scenario 4 adequately answers this requirement.

7.3.4. Stability

In Chapter 5 we precisely defined the stability of a flow control protocol. Given the state equation that describes the dynamics of the length of the queue at the bottleneck, a flow control protocol is stable if the queue length remains bounded. This is true if the eigenvalues of the discrete time state equation lie in the unit circle, or the eigenvalues of the continuous time state equation lie in the left half plane. In the chapter, we formally proved the stability of PP. We are not aware of stability proofs for any other congestion control scheme in the literature.

7.3.5. Scalability

Scaling is required along two axes: bandwidth and network size. Higher bandwidths translate to larger bandwidth-delay products, so that sources can no longer ignore the propagation delay in doing flow control. Both our deterministic and our stochastic model explicitly model the control delay, and the PP scheme, which is based on these models, is designed to work in environments with large delays. The effectiveness of PP in networks with large propagation delays is seen in the results of scenarios 7 and 8, where, even with large control delays, PP sources behave well.

As the network increases in size, far more conversations are served at each switch. How well do our schemes cope with this? Chapter 3 showed how to implement efficient data structures to buffer data from many conversations. We believe that implementing these data structures in hardware is feasible. Special purpose hardware will allow FQ to serve a large number of conversations at high speeds.

A large number of conversations also means that the service rate fluctuations for any single conversation are smoothed out. PP is specifically designed for this environment, and, in fact, its performance will only improve as the number of users increases. Thus, we claim that our congestion control scheme scales well along both axes.

7.3.6. Simplicity

The simplicity requirement is that a congestion control scheme be easy to specify and implement. Both FQ and PP are conceptually simple, and can each be implemented in under two pages of C code. The control law of Chapter 5 translates to a single line of code, and the entire fuzzy controller takes about 20 lines of code.

One possible complication with PP as we presented it is that is requires one timer per packet. However, this can be changed to a single timer per conversation with little loss of performance. As a packet is transmitted, this timer is set to the packet's timeout value. When the timer expires, the last unacknowledged packet is retransmitted. With this scheme, the timeout interval is a little larger than the correct value, but there is a substantial savings in implementation cost.

FQ conceptually requires per-packet queueing, but, as we saw in Chapter 3, this can be implemented by a single ordered linked list. In any case, with hardware support, we expect that even per-conversation queueing can be implemented at high speeds (we are aware of one implementation that enqueues and dequeues ATM cells at 1.2 Gbps).

These figures, however, ignore the realities of protocol implementation, particularly in the Unix kernel. We are aware of the complexity of implementing a protocol at the kernel level, but there is nothing specific to PP or FQ that makes it any harder to implement in the kernel than any other equivalent protocol.

7.3.7. Fairness

FQ, as its name signifies, makes an effort to deliver min-max fairness to all the conversations that it serves. Fairness, however, requires that the flow control protocol be slightly sophisticated about managing its buffers. We saw that the generic flow control protocol does not allow for fair bandwidth allocation even with FQ. However, in all the simulated scenarios, the PP/FQ protocol pair provides fair (or nearly fair) bandwidth allocations to the sources. Thus, we claim that PP/FQ is a fair congestion control scheme.

We conclude that our schemes, with some minor caveats, satisfy the requirements raised in the first chapter. This is not true for a majority of the congestion control schemes in the literature.

7.4. Comments on design methodology

The success of our schemes owes mainly to our design methodology. We believe that, while one should not ignore practical problems, congestion control schemes must be based on a sound theoretical basis. This section presents the mapping from theory to practice that underlies our design effort.

Fair queueing is based on the principle of min-max fairness. The definition of min-max fairness immediately points to a bit-by-bit-round-robin (BR) scheme as a mechanism to obtain it, and FQ can be considered to be a practical implementation of this impractical ideal. This theoretical background makes FQ robust in the face of ill-behaved users.

The similarity of a BR scheme to time-division multiplexing motivates the deterministic model of Chapter 4. Once the model is stated, a little insight yields the packet-pair probe (this approach to the packet-pair scheme was first introduced by Prof. S. Singh and Prof. A. Agrawala (129)). This gives us a clean way to probe network state.

Having recognized that the network state may change, we see the need for a formal basis to express the dynamics of the network state. This basis is provided by control theory, and the control law used by PP is derived from a straightforward application of principles of predictive control.

The need for state estimation to implement control laws is a well known problem (49). While Kalman estimators, and other related estimators, are theoretically adequate, it is increasingly recognized that, for a large class of practical applications, fuzzy logic has an important role to play. We use some fundamental principles of fuzzy control to build the fuzzy predictor described in Chapter 5. As the simulations in Chapter 6 show, this grounding in theory greatly helps the protocols in practice.

7.5. Contributions

This dissertation has made a number of contributions to the area of reactive congestion control. We review some of them in this section.

Our main contribution lies in the design of the Fair Queueing discipline and the Packet-Pair flow control protocol. (The design of FQ was joint work with S. Shenker and A. Demers.) We believe that both schemes, though conceptually simple, have several interesting features that will make them suitable for networks of the future. The two work together to provide better congestion control than some other schemes which are widely implemented in current networks.

Besides the schemes themselves, our other contributions are in developing deterministic and stochastic models for a conversation in a network, developing a control-theoretic approach to flow control in networks of FQ servers, and design of a fuzzy prediction algorithm.

Our use of deterministic modeling (which is joint work with Prof. S. Singh and Prof. A. Agrawala) makes the exact analysis of transient queueing phenomena possible. Though the model is naive, the stochastic version of the model allows for a formal control theoretic approach to flow control more easily than an equivalent stochastic queueing model.

Control theoretic approaches to flow control have been studied earlier for single M/M/1 servers and for Jacksonian networks. Our contribution lies in carrying out the analysis for a deterministic queueing model with propagation delays. The resulting protocol was shown to be stable. Further, we have implemented the resulting protocol in a realistic network simulator, and have done extensive simulations to study its behavior in a variety of benchmark scenarios.

We recognize the importance of state estimation in a distributed system with propagation delays. While we did derive the optimal Kalman estimator for the system state, we feel that such an approach is impractical for flow control protocols. Instead, our fuzzy prediction technique provides a practical alternative that performs well, and requires no additional information from the system.

7.6. Weaknesses and areas for future work

While we believe that our work has several claims to success, there are some weaknesses as well. Our theoretical models, though adequate for our purposes, are rather naive. We believe that even better results can be achieved by using more sophisticated models for the analysis. Thus, this thesis is only a small step in providing practical solutions to congestion control. What is heartening is that, even with these naive models, much can still be achieved.

Second, the simulations in Chapter 6, though extensive, are still far from a complete study of protocol behavior. While our choice of benchmark scenarios tries to test for several aspects of congestion control schemes, there are surely other aspects that we have overlooked. This is an area where much additional work is needed.

Finally, this thesis ignores two major dimensions of congestion control. To begin with, we do not consider predictive control. However, this is considered in detail in a contemporaneous thesis by D. Verma (141). Also, we have ignored congestion control on time scales larger than multiple RTTs. Studying issues on larger time scales, and integrating the solutions into a single control scheme, is an avenue for future work. Other areas for future work are mentioned at the end of each chapter.

In conclusion, we feel that the area of congestion control is vast, and still in its infancy. We hope that this thesis makes a contribution to the field.

(132). This problem was recognized quite early (22), references (44,110) provide a detailed review of these. control scheme (117). Control Protocol (TCP) (21,108) and to recover from it (63). and interactive video (35,154). endpoints, for example, two BSD sockets (21,23). *Virtual Circuits* (VCs) (such as in Datakit (41)). not make such demands (35). (68), increasing the load on the bottleneck (101). (63,114). standstill (132). The first definition is used in references (65,117), the second in reference (63), and the third in reference (85). over the resources (140). (34,47). (35,68). in a common pool (62). a day (13,105). data transfer (20,155), peak hours than during off-peak hours (7,111,130). the recent past (32,146), For example, the SPAWN system (146) users to declare their resource needs accurately (20). (80). been extensively studied by Sanders (121-123). (29,77,92). (29), determine the effect of its sending rate on the network (117). (54,63,96,117). (44), (51,52,104), (1,38,65,79,133,137,138). (63,117), (127,137,138). Examples are the Virtual Clock scheme (154), Delay-EDD (34) and Stop-and-Go queueing (48). game (124). (6), Sanders (121-123), Douligeris and Majumdar (25-27) and Lazar *et al* (59). several orders of magnitude larger (84). and perhaps have catastrophic effects (101,132). (112) themselves overload the congested network. of the recent past (45000 kbps to 56 kbps) (69). network allocates its resources (23). (43,55,106,107,116,139,149). criterion to adopt (44). Nevertheless, it is necessary to negotiation and enforcement mechanism (20,155). e.g. (63). the switches (117). Still others assume complete control but no control over source traffic (54). having hardware support for rapid switching (100) optimizing for the average case (18) removing signaling information from the data path (41) choice of scheduling algorithm (157) an efficient call processing architecture (131). peer entities (for example, (117)) For example, a Fair Queueing (23) switch has the responsibility for (65)). very bursty (52,104). and high bandwidth off-line bulk data transfer (12). Gerla and Kleinrock (44) and Pouzin (110) transient spikes in queue occupancy (115,117). (as in CIGALE or Cyclades) (61,88). and possible congestion (65). not acknowledged 'in time' (108). When the alarm goes off, congestion is suspected (153). the switch sends information in packet headers (116) or Source Quench packets (112), choke packets (88), state-exchange packets (79,120), rate-control messages (148), or throttle packets (70) probe values (76), retransmission timers (153), throughput monitoring (147), or delay monitoring (65) such as a hardware problem (153). (14,110). The Loss-load curve scheme (148) and the selective DECbit scheme (116) seek to attention in the literature, and is surveyed in (44). Other schemes, such as Input Buffer Limit (81), Channel Queue Limit (70), Buffer Class (45) and Matsumoto's X.75 proposal (91) packet switches (100). reservation-oriented networks (157). Some exceptions are in the DEC scheme (116), the Loss-load curve approach (148) and the Random dropping scheme (90). scheme does not reduce congestion (90). (63,88). either using a window adjustment scheme (117) or a rate adjustment scheme (17,76). of the future, such cooperation can no longer be assumed (118). For example, in the DECbit scheme (117), The same is true for the Jacobson-Karels TCP scheme (63). (90), and completely by the Loss-load curve method (148). developed for the Datakit network (41). This is the approach taken by Zhang in the Flow Network (154), (34), by Topolcic *et al* in the ST-II proposal (134) and by Cidon *et al* for the PARIS network (16). In a hybrid scheme described in reference (24) The two-window scheme proposed by Hahne *et al* (54) this requirement can be reduced even further (96). user to obey some traffic profile (37). at the time of call set-up, and doing admission control (34). (34). (35). their data can be dropped or violation-tagged (5). (87,128,136,141). In the Flow network, the Virtual Clock mechanism is used (154), distributed flow control (34). Ferrari and Verma (36) and Kanakia (68) have proposed mechanisms based on calendar queues (11) that have a similar effect. though this was not studied in any depth (113). (46,48,68). (34,142). performance guarantee (83). is also relevant (19,31). simulator (73). congestion (44). even if a universal standard such as ISO (12) Nagle (101) proposed a *fair queueing* (FQ) fairly (55,72). Recently Morgan (98) has examined the role such queueing and has been analyzed extensively (42,86). Similar work has been presented by Zhang (154); the fair queueing algorithm presented here (157). The max-min fairness criterion (43,55,116) (116). (this emulation discrepancy bound is proved in reference (50)). The IP protocol (109) does have a field for type-of-service, Recent work (57) raises questions about the { 1~+~{1 over {N}} ({k sup 2} ~+~ k(2 epsilon -1)) right }~~for~~{P sub F} left { 1~+~{1 over {N}} ({k sup 2} ~+~ k(2 epsilon -1)) right } found in reference (23). There are few analytical results here, but Hahne (55) is crucial in the design of switch algorithms (101). queueing algorithms to do so (124). Prue and Postel (113) (94,95) the packet to be dropped is chosen at random (90). (40,56,125,154). (154). at 1.2 Gbps (69). (50,57,89,93,124,127). such as NetBlt (17), and the technology for this is well known (66). (93). since N_{ac} is strictly non-increasing in (t_{chk}, t). has been proposed (57). (58,78). traffic in the sense of (57), TCP protocol (108) with the modifications made by Jacobson (63). we used the REAL network simulator (73) Along with the work done by McKenney (93) and Heybey et al (57), this work provides the practitioner Queueing, such as the Virtual Clock algorithm (154). `i_min[k]` is the position in `max` of the swap(min(a), min(b)); swap(i_max(i_min(a)), i_max(i_min(b))); swap(i_min(a), i_min(b)); pointer consistency, i.e. that `i_min[i_max[a]] = a` and `i_max[i_min[a]] = a`. `i_min[n] = i_max[n] = n` . min(ptr) = data; for (; (ptr/2 >= 1) &&(min(ptr) < min (ptr/2)); ptr /=2) min(1) = INFINITY; save = i_min(1); max(save) = -1; if (min(a) < min(a/2)) smaller = a/2; smaller_son = (min(lson(a)) < min(rson(a))) ? lson(a) : rson(a); if (min(smaller_son) < min(a)) smaller = smaller_son; attendant inefficiencies (50). of Bessel functions (135); of data traffic in current networks (13,51), (129). (144,145). analyzed in (129). (143,145). flow control scheme (17). scheme modified an idea described by Clark et al. for NETBLT (17), and sent to the source using a state exchange protocol (120). (64,97). so, unlike TCP (63,153), (64). (T_0, T_1) if in every time interval $(t_1, t_2) \in (T_0, T_1)$, $(T_0 + 2R(0), T_1)$. conditions are trivially satisfied in $(T_0 + 2R(0), T_1)$. For example, the Jacobson-Karels version of TCP (63) and so will lose bandwidth when V increases (64). Jain's delay based scheme (65) will respond (129). In recent work (157), we have shown that the FQ discipline is similar to the Virtual Clock (154) and Delay-EDD (34) service disciplines. though not using packet pairs, is described in reference (53), (63,108) or Jain's delay-based approach (65). As mentioned in (2), Separation Theorem (3). In recent work, Mitra et al (96,97) Processor Sharing (50), (13,51). (B/2 mi nb (k) mi sk pl mu (k) RTT(k) pl mu (k+1) RTT (k + 1)) (B/2 mi n hat sub b (k) mi sk pl mu hat (k) RTT(k) pl mu hat (k+1) RTT hat (k + 1)) (B/2 mi n hat sub b (k) mi sk pl 2 mu hat (k) RTT(k)) (B/2 mi n hat sub b (k) mi sk pl 2 mu hat (k) RTT(k)) (B/2 mi n hat sub b (k) mi sk pl 2 mu hat (k) RTT(k)) Similar studies are mentioned in (30). chosen to lie inside the range (0,B), such as the second method of Liapunov ((102) page 558). known and robust technique (49). bold "x" eq left (pile { nb ˜ above mu ˜ above mu sub -1 ˜} right) $\mathbf{G} = \begin{bmatrix} 1 & -Rtt & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ $\mathbf{H} = \begin{bmatrix} Rtt \\ 0 \\ 0 \end{bmatrix}$ $\nu_1 = \begin{bmatrix} 0 \\ \omega \\ 0 \end{bmatrix}$ $\mathbf{C} = (\,0\ 0\ 1\,)$ $E((\hat{x}(t) - x(t))^T(\hat{x}(t) - x(t)))$, to yield many other types of filters (49). right) (nu sub 1 sup T (theta) nu sub 2 (theta)) ˜ right } eq left (pile { { Q ˜ S˜ } above { S sup T ˜ R˜ } } right) ˜delta $\mathbf{Q} = E\begin{bmatrix} 0 & 0 & 0 \\ 0 & \omega^2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $\mathbf{S} = E\begin{bmatrix} 0 \\ \omega\nu_2 \\ 0 \end{bmatrix}$ system. We follow the derivation in (49) (pg 249). bold "K" (k) eq (bold "G" SIGMA (k) bold "C" sup T pl bold " S ") (bold (82,152,158). (151), found in reference (75). We also make the fluid approximation (1), This approximation is commonly used (8,133), and both analysis (97) and simulations show time interval (t, t+J), but in fact can work together (67). scheme proposed by Hahne *et al* (54) is attractive. (6,9,10,27,127). work by Jain (115) and Jacobson (63), In recent work, Ko *et al* (79) described originally by Agnew (1), and since extended by Filipiak (38) and Tipper *et al* (133). Vakil, Hsiao and Lazar (138) Vakil and Lazar (137) have considered Robertazzi and Lazar (119) and Hsiao and Lazar (60) in the benchmark suite described in reference (23) is comparable to that of the DECbit scheme (117), (63,117)), messy and only partially understood (156). experimental high speed network testbed (69). lk1 eq 1 over { RTT(k)} (B/2 mi nhat-1 mi sk pl mu RTT(k) pl mu RTT (k)) lk1 eq 2 mu mi lk pl 1 over { RTT(k)} (B/2 mi nhat-1) lambda (2) eq 2 mu mi mu pl 1 over { RTT (2)} (B/2 mi B/2) lambda (3) eq 2 mu mi mu pl 1 over { RTT(3)} (B/2 - B/2) We follow the description of LQG presented in (102) (pg. 835). bold "Q" eq left (pile { { 1 ˜0 ˜0 ˜ } above { 0 ˜0 ˜0 ˜ } above { 0 ˜0 ˜0 ˜ } } right) (99) (93,117,154), (33). as in reference (12). traffic (13,51,105). (33,103), (4,28) (73,74). transport flow control protocol (108,150), (63,126,154,156), the DECbit scheme (114-117). as implemented in XNS's SPP (150) or in TCP (before 4.3 Tahoe BSD) (108), Jacobson and Karels' (JK) modified TCP (63) (15,115-117), (63,71)). protocols such as NETBLT (17) congestion avoidance scheme (65). (116,148), (see reference (116) and the simulation results below), on the description in references (115,116). of 40 and 570 bytes in the Internet (12). Floyd and Jacobson in (39,40)). (39,40). (54). (though global tradeoffs are still possible (6)). and has also been studied exhaustively in references (39,40). (101)). by Prof. S. Singh and Prof. A. Agrawala (129)). laws is a well known problem (49). thesis by D. Verma (141).