

UNIVERSITE DE NICE-SOPHIA ANTIPOLIS - UFR SCIENCES

Ecole Doctorale STIC

THESE

Présentée pour obtenir le titre de

Docteur en SCIENCES

de l'Université de Nice Sophia Antipolis

Spécialité : Informatique

par

Naceur MALOUCH

titre

Modélisation et optimisation de mécanismes de services à valeur ajoutée dans Internet

Soutenue publiquement le 06 Janvier 2003 devant le jury composé de :

M.	Serge	Fdida	Université Pierre et Marie Curie - France	Rapporteur
M.	Alain	Jean-Marie	Université Montpellier II - France	Rapporteur
M.	Zhen	Liu	IBM T. J. Watson - USA	Directeur
M.	Philippe	Nain	INRIA Sophia-Antipolis - France	Examineur
M.	Michel	Riveill	Université de Nice Sophia-Antipolis - France	Président
M.	Zhi-Li	Zhang	Université de Minnesota - USA	Rapporteur

Contents

1	Introduction	1
1.1	Research Topics in Quality of Service in the Internet	1
1.2	Key Contributions	3
1.3	Thesis Organization	4
I	Differentiated Services	5
2	Overview of Differentiated Services	7
2.1	Marking and Per-Hop Behavior	8
2.2	Traffic Conditioning	8
2.3	Implementing Traffic Conditioning	9
2.3.1	Time Sliding Window	9
2.3.2	Token Bucket	9
2.4	Implementing Per-Hop Behaviors	10
2.4.1	Scheduling	11
2.4.2	Queue Management	12
2.5	Example of Services	13
2.5.1	Expedited Forwarding	13
2.5.2	Assured Forwarding	13
2.6	Quantitative Evaluation of DiffServ	14
2.6.1	Service Performance Metrics	14
2.6.2	Evaluation of EF Service	15
2.6.3	Evaluation of AF Service	16
2.7	Conclusion	17
3	TCP Throughput in Networks with Differentiated Services	19
3.1	TCP in a Nutshell	19
3.1.1	Slow Start	20
3.1.2	Congestion Avoidance	20
3.1.3	Fast Retransmit	20

3.1.4	Fast Recovery	20
3.2	TCP Throughput Analysis	21
3.2.1	Network Assumptions	21
3.2.2	Relationship between Throughput and Window Size of TCP	22
3.2.3	Relation between Congestion Avoidance Area and Loss Probabilities	23
3.3	TCP Window Size	23
3.3.1	Triple Duplicate Events	24
3.3.2	Timeout Events	28
3.4	Throughput Model Validation	28
3.5	Conclusion	30
4	Performance Analysis of TCP with RIO Routers	33
4.1	Network Model	34
4.2	Characteristic Equations and Computational Scheme	36
4.2.1	Equations of TCP Dynamics	36
4.2.2	Equations of the RIO Router	37
4.2.3	Fixed Point Method	38
4.3	Model Validation	38
4.3.1	Fully Overlapped Case	38
4.3.2	Non-Overlapped Case	39
4.3.3	Partially Overlapped Case	39
4.4	Impact of RIO Parameters on QoS and Fairness	41
4.4.1	Throughput	41
4.4.2	Delay	42
4.4.3	Drop probability	43
4.5	Conclusion	45
II	Overlay Multicast	49
5	Overlay Multicast: A New Multicasting Approach	51
5.1	Overlay Multicast	52
5.2	Overlay Multicast Approaches	54
5.2.1	End-System Multicast	54
5.2.2	Proxy-Based Multicast	54
5.3	Overlay Multicast Design	55
5.3.1	Group Management	55
5.3.2	Overlay Optimization	55
5.4	Routing in Overlay Multicast	55
5.4.1	Bandwidth Optimization	56

5.4.2	Bandwidth Constrained Delay Optimization	57
5.4.3	Delay Constrained Bandwidth Optimization	58
5.5	Conclusion	59
6	Bounding Delay in Proxy-Assisted End-System Multicast	61
6.1	Architecture and Model	62
6.2	Uniform Edge Overlays	65
6.2.1	Minimizing Fanout-constrained Tree Depth	65
6.2.2	Applying Majorization to FCFs	67
6.2.3	Minimizing Proxy Involvement	68
6.3	Heuristics for Non-Uniform-Edge Overlays	69
6.3.1	Design from Theoretical Observations	70
6.3.2	Additional Modifications	71
6.4	Heuristic Evaluation	72
6.4.1	Experimental Setup	73
6.4.2	Performance Evaluation	74
6.5	Conclusion	76
7	Bandwidth-Sharing Schemes in End-System Multicast	79
7.1	Network Model	80
7.2	Algorithms	81
7.2.1	Algorithm CLUSTER	82
7.2.2	Algorithm DISPERSE	83
7.2.3	Algorithm Extensions	83
7.3	Homogeneous Network	85
7.4	Queuing Model: A Stochastic Knapsack	88
7.5	Simulation Results with Static Group Membership	89
7.5.1	Experiment 1: Homogeneous case	90
7.5.2	Experiment 2: Variable Node Capacities	91
7.5.3	Experiment 3: Variable Session Rates	92
7.5.4	Experiment 4: Variable Session Sizes	92
7.5.5	Reservation Policy	93
7.6	Simulations with Dynamic Group Membership	93
7.7	Conclusion	95
8	Conclusions and Future Work	97
8.1	Differentiated Services	97
8.2	Overlay Multicast	98
	Bibliography	112

Chapter 1

Introduction

Internet started its inception in the early 1960s as a part of a research project supported by the American government through its funding agencies. The objective was to develop a set of protocols to provide connectivity between heterogeneous linked packet networks. These protocols became known as the TCP/IP protocol suite: Transmission Control Protocol (TCP) and Internet Protocol (IP). IP provides a *basic service* of unreliable packet delivery: There is no guarantee that a given packet reaches its destination, any received packet could be inaccurate, and the packets may be received in the wrong order. Furthermore, Internet nodes forward packets using a simple First Come First Served (FCFS) scheduling scheme, and packets are served whenever possible which constitutes the well-known *best-effort* service. TCP is, built over IP, a reliable connection-oriented service with end-to-end flow control and congestion control functionalities. TCP, however, does not provide any guarantees on the quality of the transmitted packets such as a bounded delay, a high bit rate, or a small loss rate.

With the advent of new applications, especially the World Wide Web (WWW) that take advantage of TCP/IP efficiency, many public and private networks have joined the Internet. A recent Internet survey (July 2002) [65] estimates the number of hosts to be more than 160 million. While some applications such as electronic mail and file transfer still work well, other applications such as audio and video conferencing, shared whiteboard, telephony and television, database replication, etc, need additional communication bandwidth requirements. For example, if a congestion occurs on the path during some file transfer, then TCP will provoke a decrease in the rate of the file transfer. This may also cause an additional delay. However, TCP ensures that the file will reach the destination properly. For real-time applications such service is unacceptable. Insufficient bandwidth or large delays decrease the quality perceived by these applications.

1.1 Research Topics in Quality of Service in the Internet

A crucial question arises: How can we provide a service that is better than the current *best-effort* service ? Many new thoughts and concepts have been suggested to make new kind of applications work adequately over the Internet.

Since many new applications mentioned above have stringent requirements on bandwidth and involve the participation of more than two users, e.g. video conferencing, solutions were proposed to enhance the IP layer by adding new multicast functionalities to improve bandwidth usage. This is known as IP Multicast [36]. IP Multicast enables a source to send a single copy of a message to multiple recipients who want to receive the information. This method is more efficient than sending an individual copy to each recipient using unicast connections. A comprehensive survey can be found in [39].

However, due to difficulty of group management and access control, it is not likely to see the wide deployment of IP-supported multicast. Rather, multicast in *overlay networks* becomes a hot topic. The first deployment example of such networks is the Multicast BackBONE (MBONE) [78]. It is created using a virtual layer of tunnels over the Internet. The MBONE connects individual multicast-capable subnetworks with tunnels whose endpoints encapsulate multicast packets into normal IP packets and send them across the unicast path between the tunnel endpoints.

Another line of research to provide better than best-effort service for any application is to mimic the solution adopted for telecommunication networks which is based on resource reservation and traffic isolation. In this context, the Integrated Services (IntServ) [128, 112, 129] was proposed to provide multiple levels of delivery for applications. Two components are added to the IP layer: traffic control and signaling. Signaling is performed via a resource reservation protocol (RSVP) [129] which determines the necessary network resources that must be prepared and held in routers (switches) in order to guarantee a certain service. Traffic control implements scheduling mechanisms, *flow* classification and admission control. In the routers each flow is assigned a separate “virtual circuit” using flow-based scheduling mechanisms. The major drawback of IntServ is that routers in the core of the network must maintain state information for each flow, RSVP does not scale with the number of flows [90].

Differentiated Services (DiffServ) [70, 16] are then introduced to provide somewhat a “lightweight” version of IntServ. Instead of separating between single flows to control the traffic inside the routers, DiffServ separates between *aggregates* of flows. In the core routers, the same service is provided for flows belonging to the same aggregate. DiffServ still needs modifications in the infrastructure but less than what is required for Intserv. Since 1997, a prodigious efforts have been devoted to study DiffServ mechanisms and their impact on the performance of applications. Though the results are promising [117, 5, 99, 16, 91, 98, 111, 44], there are still some issues that prevent a wide deployment of DiffServ in the Internet such as peering and charging.

There are also many research efforts towards providing the quality of service (QoS) in the Internet, such as QoS routing [29], active networking [120], congestion control (ECN) [105], etc. In this thesis, we shall not address these issues.

On the other hand, many researchers are still of the opinion that the current Internet architecture is satisfactory, and that all additional functionalities should be implemented in the *application layer* to cope with the “end-to-end argument”. Indeed, only the application itself

can identify precisely its requirements in terms of quality of service. For example, applications that have dynamic number of users cannot ask the network for a delay bound since the total delay depends on the delays incurred in the intermediate domains which is variable. However the application can control the end-to-end delay by performing measurements and changing paths.

There has been recent work [9, 103, 31, 30, 28] on the use of specialized multicast protocols implemented in the application layer. Since it is a relatively a new approach of multicasting, the terminology used varies, such as end-system multicast, application level multicast, or more recently overlay multicast. We will use the two last terms and we will keep the term end-system multicast to refer to a category of overlay multicast.

In this thesis we will study, in two parts, differentiated services and overlay multicast. Our understanding is that both paradigms are in prospect to be deployed in the Internet. Much work has been done in these two areas, and this thesis is an effort to contribute to the previous studies.

1.2 Key Contributions

In the first part of this thesis, we summarize some of the analytical works that study two services that can be provided by a differentiated services network. We discuss their pros and cons, then we focus on the evaluation of buffer management mechanisms used for building the assured service and their impact on TCP. The assured service provides different levels of forwarding assurances for packets entering the network. We develop a general model that gives closed form expressions of the throughput of a TCP connection with different colors (levels) of packets. In particular, we compute the throughput and the window size of a TCP connection whose packets are marked with two colors by a token bucket mechanism. The input parameters are the rate of token generation, the buffer size of the token bucket, the two average loss probabilities of packets, the average round-trip time and the retransmission timeout constant. The interest of these expressions is twofold. First, they can be used to share fairly the bandwidth among TCP and UDP flows in a DiffServ network. Second, they give insights for network dimensioning as they can predict the load of TCP traffic.

We also develop an approach to analyze the performance characteristics of TCP sessions in the presence of network routers which deploy the Random Early Detection (RED) mechanism with two *in* and *out* drop probability functions (RIO). We consider the case with a large number of TCP sessions which use token buckets for marking *in* and *out* packets at the entrance of the network. We derive a set of equations that govern the evolution of these TCP sessions and the routers under consideration. We solve these equations numerically using a fixed point method. We use a novel incremental technique to avoid oscillations around the fixed point solution. Our analysis can capture characteristics of both RED and Tail Drop (TD) mechanisms in the RIO router. Various performance analyses are then carried out using this approach in order to study the impact of the RIO parameters on the performance characteristics of TCP sessions. Further, we derive preliminary configuration rules for setting these parameters.

In the second part, our contributions pertain to the overlay multicast. We start by exploring the problem of building hybrid proxy/end-system application layer multicast trees that meet fixed end-to-end delay bounds. Up-to-date works have considered only two approaches: proxy-based multicast or end-system multicast. We use simple graph theoretic network models to develop an algorithm that minimizes costs associated with the utilization of proxies while meeting a fixed delay bound. We formally prove its optimality in a fully-connected overlay network with uniform-length edges. We adapt this algorithm for the design of a heuristic and we evaluate the heuristic for simulated transit-stub networks with variable-delay edges. We compare our heuristic in a proxy-free environment to a previously developed heuristic and show that our heuristic typically yields further reductions in the maximum session end-to-end delay.

In addition to delays, optimizing bandwidth usage is a crucial issue in overlay multicast because multicast sessions are often expected to compete for the same network resources. However, most protocols to-date assume implicitly that they have isolated access to the bandwidth. We proposed two new algorithms that aim to maximize the number of multicast sessions that can coexist in the same network. We try to prove analytically that when node capacities as well as session requirements are identical, a clustering algorithm is optimal. In a more heterogeneous environment, we use simulations to investigate the impact of the algorithms on the blocking probability in various scenarios including dynamic group membership. We also model the heterogeneous environment as a queueing system and use this system to derive a theoretical lower bound on the blocking probability that applies to any solution in the area.

These results can be used to help future development of overlay protocols that partition resources among multiple sessions.

1.3 Thesis Organization

The rest of this dissertation is organized in two parts. The first part is dedicated to the differentiated services paradigm. The second part examines the new multicasting approach using overlays. Accordingly, in Chapter 2, we present an overview of differentiated services mechanisms and their evaluation. In Chapter 3 we present a detailed analysis for TCP throughput in a network with different priority levels. In Chapter 4, we analyze the performance of TCP in interaction with RIO routers.

In Chapter 5 we present the overlay multicast approach and discuss the optimization problems involved. In Chapter 6, we introduce a formal description of overlay multicast through a set of definitions and theorems some of which will also serve for Chapter 7. Then, we focus on optimization problems in a hybrid proxy/end-system environment. In Chapter 7, we study bandwidth sharing schemes in end-system multicast. Finally, in Chapter 8, we present the conclusions and point out further research directions.

Part I

Differentiated Services

Chapter 2

Overview of Differentiated Services

To avoid the scalability problems inherent in the Integrated Services architecture [90] the research community has developed a new set of protocols that push most of the complexity and state information to the network edges. To do so, it is obvious that the core router can not maintain information for each flow traversing it. The main idea behind the Differentiated Services (DiffServ) is to discriminate between *packets* instead of *flows* in contrast with IntServ. The applications are divided into a relatively small number of classes. A class can gather applications that require the same quality of service. Thereafter, once a packet enters the network, it is assigned to the appropriate class. The network nodes are classified into two types: edge routers and core routers. The core routers does not maintain any state information to forward the packets according to their classes. Since the number of flows in the edge of the network is relatively small, the edge routers can perform per-flow (and/or per-class) actions.

Since 1997, many schemes have been proposed to provide differentiated services for the Internet. The most popular one is the "Two-bit Differentiated Services Architecture" suggested by Jacobson *et al.* [70] which describe two services, the premium service [67] and the assured service [32]. Afterwards, the DiffServ working Group of the IETF standardized and recommended a set of schemes that are influenced by the previous proposals [70, 34, 4, 47, 125].

Defining the schemes and the protocols is not sufficient. A theoretical study must also be conducted to analyze the behavior of the involved mechanisms and their impact on the performance of the applications and also the transport protocols. Many questions arise such as to what extent we can differentiate between packets and can we control the quality of service offered to a flow by controlling the quality offered to the aggregation to which the flow belongs.

In this chapter, we present the different components of the DiffServ architecture. Then, we describe two popular services defined by the IETF, the assured forwarding service (assured service) and the expedited forwarding service (premium service). In section 2.6, we discuss the quantitative evaluation of these services in the literature.

2.1 Marking and Per-Hop Behavior

To assign a packet to a class of service, the packet is marked using a special field in the packet IP header called Differentiated Services Code Point (DSCP). The marking is performed at the edge routers. Using the terminology defined by the IETF [99] a class of service is translated into a Per-Hop Behavior (PHB) in a core router. A PHB specifies the way the packet is forwarded. Within each DiffServ capable router there is a mapping between DSCP values and PHBs. Every ISP can create and map his own PHBs in his DiffServ domain to provide services for network users. Usually a complete service includes a set of PHBs called PHB group.

2.2 Traffic Conditioning

Traffic conditioning is performed at the ingress node of the DiffServ domain. It must determine the required behavior for each packet of a traffic flow entering the domain and then decides whether to mark, delay, or drop the packet. The traffic flows are categorized by means of classifiers. The classification matches the received packets to some profile and then passes the packets to the conditioner associated with that traffic. The DiffServ framework allows a traffic classifier to run in two modes: Multi-Field (MF) mode or Behavior Aggregate (BA) mode. In the MF mode, packets are classified based on the content of some arbitrary number of header fields including all the information at the flow level and the application level such as the source address, the destination address, the source port, the destination port, the protocol number and the DSCP field. In the BA mode, only the DSCP is used to classify the packets. Thereby, packets belonging to different flows will pass through the same traffic conditioner and receive the same treatment.

A traffic conditioner consists of a meter combined with a policer. A policer could be a marker, a dropper or a shaper.

Meter

The meter measures the temporal properties (e.g. rate, burstiness) of a traffic selected by a classifier and check its conformity to the traffic profile defined in the Service Level Agreement (SLA) which is the negotiated contract between the DiffServ provider and the customer. The PHB of the packet is determined according to the level of conformity. The PHB might be specified directly in the traffic profile. We should notice that an SLA could be implemented statically (changes manually weekly, monthly or annually) or dynamically using bandwidth brokers [70]

Marker

Once the PHB is known the packet is marked with the corresponding DSCP. Usually, the marker is implemented together with the meter and it is called simply marker, e.g. [61, 62, 43, 18, 48].

Shaper

The shaper delays the packets in a traffic in order to force them into compliance with the traffic profile.

Dropper

The dropper simply drop the packets that do not conform with the traffic profile.

2.3 Implementing Traffic Conditioning

The notion of traffic conditioning has already existed before the differentiated services. Most of the implementations developed to measure and shape the traffic for Intserv and ATM networks could be adopted for DiffServ. In the following, we present two implementations that were introduced with DiffServ networks.

2.3.1 Time Sliding Window

The Time Sliding Window (TSW) is a technique to estimate the average rate of a traffic stream's arrival rate. It provides a smooth estimate of the average rate over a period of time (window). Upon each packet arrival, TSW computes the average rate using the packet size of the arriving packet, the last inter-packet arrival time the number of bytes sent in the previous time window and the window length. If the window length is very small, the estimated rate represents the instantaneous rate. if the window length is very large, then the estimated rate converges to the average (non smoothed) rate of the stream. TSW is sketched by algorithm 1 [33].

Algorithm 1: Time Sliding Window Estimator

Input: A window length win_len

Output: The estimated average rate

TSW(win_len)

- (1) $bytes_in_TSW = avg_rate * win_len$
- (2) $new_bytes = bytes_in_TSW + PKTSIZE(p)$
- (3) $avg_rate = new_bytes / (now - T_front + win_len)$
- (4) $T_front = now$

The rate estimated by TSW is compared against the rates specified in the traffic profile and the packets are marked accordingly to form a complete traffic conditioner. An example of marking policy is presented in [43].

2.3.2 Token Bucket

The Token Bucket (TB) mechanism is a popular implementation of a meter. A TB compares the rate of incoming packets to a given rate. Figure 2.1 illustrates the TB mechanism.

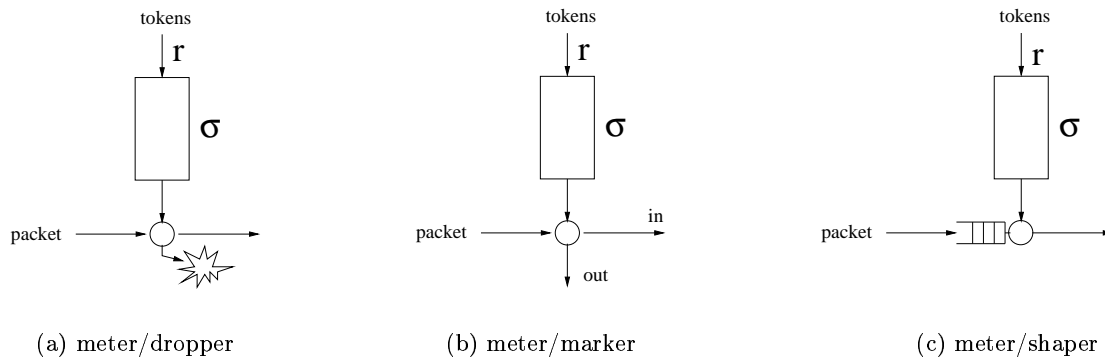


Figure 2.1: Token Bucket mechanism

Tokens are generated at (constant) rate r and are stored in the bucket of size σ . Newly generated tokens are dropped if overflow occurs at the bucket. When a packet arrives, if there is at least one token in the bucket, the packet consumes the token and leaves the TB¹. In this case the packet is in agreement with the traffic profile represented by the TB parameters (r, σ) . If the bucket is empty, then one action among the following could be taken:

- the packet is dropped (Figure 2.1(a))
- the packet is marked (Figure 2.1(b))
- the packet waits in a buffer until at least one token is available (the packet is shaped, Figure 2.1(c)).

Note that a token bucket itself has no discard or priority policy even if it is usually associated with marking. If the TB is used as a shaper, then during a time interval of t the average output rate is upper bounded by $r + \frac{\sigma}{t}$. r is interpreted as the peak rate of the input traffic and σ is the burst size tolerance. In other words, the maximum output rate corresponds to the token generation rate in average (when $t \rightarrow +\infty$).

2.4 Implementing Per-Hop Behaviors

As we mention in the previous section a PHB is responsible for forwarding the packets with different DSCPs. The mechanisms that implement PHBs in the core routers must be able to discriminate between packets that have different marking. They must provide tools to control the performance metrics of the transmission: the delay, the rate and the drop probability to create different *behaviors*. Usually, this is handled by a queueing system attached to the outgoing interface of the router. A queueing system includes a scheduling mechanism and a queue

¹In practice, the tokens are accounted in bytes, in that case the packet leaves the TB when the total number of bytes in the bucket is greater than the packet size in bytes

management mechanism. The traditional queueing system deployed in best-effort networks is a FIFO (First in First Out) queue with a Tail Drop (TD) policy. The TD is also suggested recently to be replaced by Random Early Detection (RED): the congestion control mechanism.

To provide differentiated services, a scheduling mechanism needs to be designed with several priority classes. A scheduler must control access to the link bandwidth for each class. The queue management mechanism must adopt a per-class drop policy.

2.4.1 Scheduling

Several scheduling mechanisms were proposed for various purposes. One queue can be used for each defined class of service (DSCP) or one queue for all the classes. We present below some of these mechanisms which were used in a DiffServ context.

Priority Queueing (PQ):

With this policy, the packet with the highest priority is always selected first. A packet of lower priority is selected only if there is no packets from all higher priorities in the queueing system. Usually one virtual queue is used for each priority level, but the use of one shared queue for all type of packets is also possible. This policy can result in starvation of lower priority packets if the higher priority traffic intensity is larger than the capacity of the scheduler.

Weighted Fair Queueing (WFQ):

Introduced in [37] as a packet by packet version of the Generalized Processor Sharing scheduler. A queue is assigned for each class of packets ². A weight is attributed to each queue so as it will receive a different share of the bandwidth. The next packet to send in WFQ is the one with the smallest departure time computed using a hypothetical bit-by-bit round robin service discipline. This method ensures that even with different packet sizes, the scheduler approaches an absolute fair bandwidth allocation. WFQ provides a virtual separation between the packets of each queue which means also a protection against ill-behaved classes [37]. Similar scheduler or variants of WFQ were also developed such as Worst case fair Weighted Fair Queueing (WF²Q) [13], Worst case fair Weighted Fair Queueing+ (WF²Q+) [66], Virtual Clock (VC) [132], Weighted Round Robin (WRR) [57].

Class Based Queueing (CBQ):

CBQ is based on the concept of hierarchical link-sharing. The output link is shared among classes using a hierarchical tree. Each node in the tree corresponds to a class of traffic. Only a leaf class is associated with a queue in the scheduler. Non-leaf classes represent aggregation of classes which are related, for example, they have similar performance requirements or belong to the same user. Each class has a bandwidth allocation and a priority. CBQ

²WFQ was introduced to support one *flow* per queue, but it can be also applied for aggregates i.e. *classes*.

serves classes with the same priority in a round robin fashion [51]. H-WFQ and H-WF²Q are other hierarchical link-sharing schedulers and they are explored in [66].

Waiting Time Priority (WTP):

This is a priority scheduler in which the priority of a packet increases proportionally with its waiting time [40]. The next packet to send is the one with the highest instantaneous priority. Since the priorities change over time, the bandwidth is distributed *dynamically* among the classes. The priority of the head of each queue reflects the load of the queue in the recent past. WTP can provide different packet delays for each class. This scheduler was first studied in [76] under the name Time Dependent Priorities.

2.4.2 Queue Management

Another set of mechanisms that can be adjusted to provide differentiation is queue management. In a best-effort packet-switched network, buffers are designed to absorb burst arrivals of packets in order to reduce losses. Naturally, packets are dropped when the queueing delay increases significantly or when the buffer overflows. Furthermore, The Internet Research Task Force (IRTF) recommends the use of additional dropping mechanisms in the routers to complement end-to-end control mechanisms [19]. These new dropping mechanisms is referred to as Active Queue management (AQM). AQM aims to (i) reduce the number of dropped packets in the routers, (ii) reduce the queueing delays, and (iii) increase the buffer availability.

The most popular AQM for the Internet is Random Early Detection (RED) introduced by Floyd and Jacobson in [52]. RED algorithm operates as follows: for each incoming packet, RED computes an Exponentially Weighted Moving Average (EWMA) of the queue length, then it computes a dropping probability using a preset function (Figure 2.2). The packet is dropped randomly with respect to the obtained probability. This method is supposed to be beneficial to responsive flows that react to packet losses by reducing their transmission rate because it prevents from the so-called *global synchronization*. Since losses are random, flows will react at different periods and bursts originated from different flows will not enter the queue simultaneously causing grouped losses.

Many variants of RED have also been proposed, e.g. Stabilized RED (SRED) [100], Adapted RED (ARED) [46, 50], and Flow RED (FRED) [81]. Other proposals for AQM were also developed such as Blue [45] which uses packet loss and link utilization history to compute the dropping probability instead of the average queue size.

Normally, any AQM which do not maintain per-flow state information, can be extended to cope with DiffServ networks since it provides a tool to control the packets entering the queue. The key idea is to drop differently packets belonging to different classes even if they share a common queue. For example, we can extend RED for multiple classes of service by assigning different drop probability functions to each class (Figure 2.3). For example RIO (RED with In and Out) which is introduced in [33] defines two classes of packets: *in-profile* and *out-of-profile*.

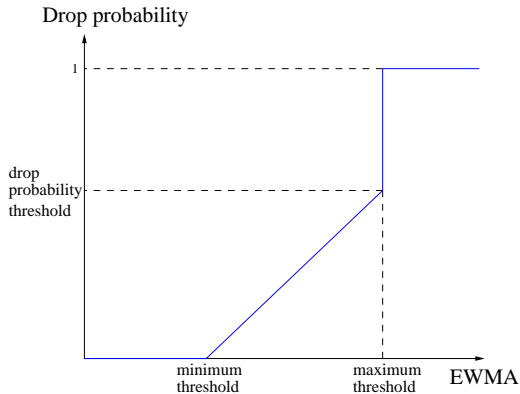


Figure 2.2: A piecewise linear function for RED

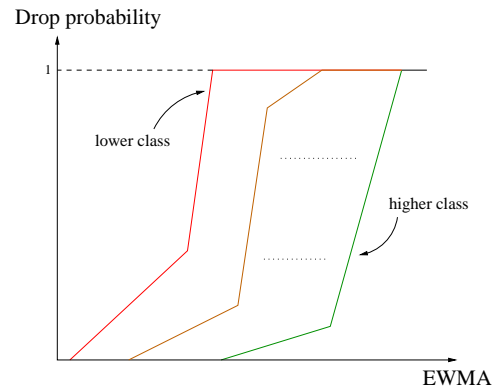


Figure 2.3: RED with multiple classes

In chapter 4 we will analyze more in detail this mechanism.

2.5 Example of Services

The combination of a set of traffic conditioning policies and a set of Per-Hop Behaviors define a differentiated service. Various services can be built in a DiffServ domain. In the following we present the two popular services that were defined by the DiffServ Working Group within the IETF.

2.5.1 Expedited Forwarding

The primary goal of this service is to provide an equivalent leased line service between two end-systems. It is also useful for some applications with strict QoS requirements such as Voice over IP and real-time video. It provides a minimal assured rate, low latency, low loss, and low jitter service. Loss, latency and jitter are all due to the queues traffic experiences while transiting the network [69]. Therefore to provide the EF service, a traffic aggregate must see no or very small queues which means that the maximum input rate at any time must be no greater than the minimum output rate. Various scheduling mechanisms could be used to achieve this service such as PQ, WFQ or CBQ since they can allocate a fixed share of the bandwidth to the EF traffic and then the ability to configure the output rate. A Tail Drop queue with a small size is used to buffer EF traffic. RED can also be used to spread randomly losses among the flows of aggregates. A shaper at the network edge is usually necessary to bound the rate of EF traffic in order to avoid starvation of other traffic.

2.5.2 Assured Forwarding

Assured Forwarding (AF) defines N classes of service each of which defines M different levels of drop precedence. Currently, the standard track specified in [60] recommends $N = 4$ and $M = 3$ for general use. Each AF class receives independently of the others a certain amount

of resources i.e. bandwidth and buffer space. Within each class the lower the level of drop precedence the greater the delivery probability in times of congestion. Packets that conform to the traffic profile (*in-profile*) while entering the network are assigned the lower level of drop precedence. AF can provide relative differentiation of services by allocating a large amount of forwarding resources to some classes [40]. A well known example is the Olympic service model which consists of the Gold, Silver and Bronze classes [60, 40]. By subscribing to one AF class, a customer will subscribe to an *expected capacity* rather than a *guaranteed capacity* [33]. This is because *in-profile* traffic is multiplexed with *out-of-profile* traffic in the same buffer, in times of congestion the amount of dropped packets as well as the available bandwidth is uncertain. Most of the scheduling mechanisms such as WFQ, CBQ and WTP can implement the AF service. To implement the levels of drop precedence, we can use RED with multiple classes of service. At the edge of the network, traffic conditioning can be performed via cascaded Token Buckets.

2.6 Quantitative Evaluation of DiffServ

In this section we turn our attention to the efforts that aimed to evaluate the performance of DiffServ mechanisms. Most of the works that have been done to evaluate the proposed services used simulations and/or experimentations. A quantitative study is required to obtain closed form expressions or expressions that can be solved numerically using a simple and fast algorithms. While they involve simplifying assumptions, they provide better understanding of systems and they are the more practical way to solve optimization problems and thus derive dimensioning and configuration rules. Furthermore, they can answer on-line (and off-line) whether an SLA is achievable or not. First we describe the performance metrics that can be used to study *quantitatively* a service, then we discuss the main results that have appeared in the literature.

2.6.1 Service Performance Metrics

There are three performance metrics that can be used to study the level of differentiation between aggregates and also to compare between different approaches that provide the same service. These are: throughput, latency and drop probability.

Throughput The throughput is the rate at which data (packets) is transmitted including losses.

It is an important measure of performance for TCP applications.

Latency It is controlled by the queuing delays incurred by the routers and the propagation delays. It is possible to control the delay by scheduling mechanisms.

Drop probability It is determined by the percentage of total dropped packets. In an AF class, this measure is crucial because the differentiation between the precedence levels results from providing different drop probabilities for each level.

2.6.2 Evaluation of EF Service

Priority Scheduling (PS) is usually used as the router implementation to evaluate the performance of EF. In [111], the expressions of the expected delay and the loss probability are derived in the case of two traffic classes: preferred and non-preferred (EF traffic and non EF traffic, respectively). The network is modeled by a system of two Tail Drop queues served according to a pre-emptive Priority Scheduler. The preferred and non-preferred traffic is assumed to be two independent Poisson processes and the packet transmission times are assumed to be exponentially distributed. This simplified system is analyzed via a two-dimensional Markov chain representing the number of preferred and non-preferred packets in the system. Then the Markov chain is solved numerically. May *et al.* [91] studied the special case where the buffer size of preferred packets is finite and the one of non-preferred packets is infinite. The finite buffer is used to bound the delay of preferred packets as it is one of the primary goals of EF (Figure 2.4).

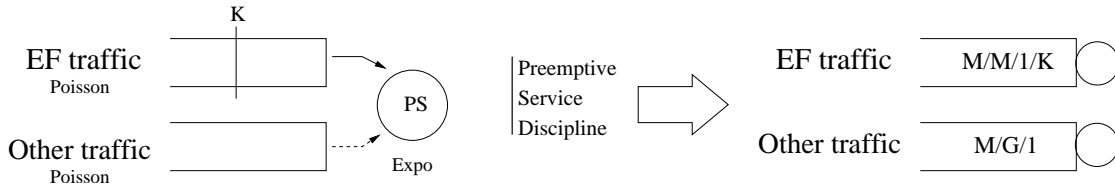


Figure 2.4: A queueing model for EF

The Closed form expressions of the expected waiting delay and the loss probability of preferred packets corresponds to those given by an M/M/1/K system. For the non-preferred class, two upper bounds are derived for the delay. The first one is obtained using simply the result derived in the case where the buffer size of preferred packets is infinite since it represents the worst scenario for non-preferred packets. The second upper bound is more tight. It is computed numerically using the expected value of the busy period in the M/M/1/K system.

Using the above models ([111] and [91]) interesting results are extracted concerning the EF behavior with respect to the traffic model (Poisson). The key observation is that the waiting time is not affected by changes in the non-preferred packet arrival rate which means that with PS, an ISP can guarantee delays for EF traffic irrespective of other traffics load. Moreover, the waiting time of preferred packets is always smaller than the waiting time of non-preferred packets. and when the total load is high (congestion) the difference becomes significant. The loss probability of preferred packets can also be guaranteed since it is thoroughly controlled by the buffer size of the preferred queue. Actually, this buffer size controls the trade-off between the loss probability of preferred packets and the waiting delay of non-preferred packets. By decreasing the buffer size, the loss probability of preferred packets is increased, but the waiting time of non-preferred packets is decreased. However, this can not avoid starvation of the non-preferred class if the load of preferred packets is greater than the capacity of the scheduler.

The analysis is extended in [111] to cover bursty EF traffic. Bursty arrivals are modeled

by N homogeneous Markov modulated On-Off sources. When a source is in the On state, it transmits fluid at a constant rate. The aggregated preferred traffic is assumed to be smaller than the capacity of the scheduler which means that a sort of source admission control is performed. The non-preferred traffic is modeled simply by a constant bit rate source. Numerical results shows that under such assumptions, preferred packets, if admitted, they incur no delays and no losses. Furthermore, the number of preferred sources that can be supported is independent of the sources' burstiness. From these results one can see that PS is suitable for applications with stringent delay and loss requirements.

2.6.3 Evaluation of AF Service

Studying the AF service is quite intricate since the quality of service offered is statistically guaranteed. Buffer management mechanisms plays a major role in the design and the implementation of each AF class. May *et al.* [91] studied analytically the impact of a simplified version of RIO (RED with In and Out) on the goodput of Poisson traffic with two classes of packets, tagged (In) and non-tagged (Out), the fraction of non-tagged packets is less than 50%. Packets are accepted into the buffer until it reaches half the buffer size, then packets are dropped with a probability that increases linearly to 10% for tagged packets and to 90% for non-tagged packets. All the packets are dropped when the buffer becomes full.

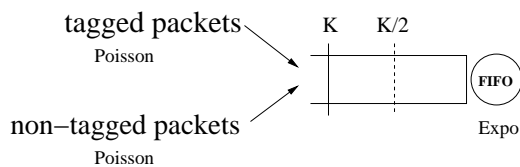


Figure 2.5: A queueing model for an AF class

The goodputs and the queueing delays with TD and with RIO are computed numerically using the stationary distribution of the number of packets in the queue. The major result is that the goodput of both tagged and non-tagged packets is little dependent on the buffer scheme when the load is low (no congestion). However, when the load is high, tagged packets get a larger fraction of the total capacity.

Kuusela *et al.* [80] used an ordinary differential equation approximation to describe the evolution of the expectations of the exponentially averaged queue lengths. They consider two classes of traffic, each of which is a Poisson stream with a time varying rate. Essentially, the differential equation is derived by considering a continuous time version of the EWMA algorithm. The instantaneous expectation of the queue length is approximated by the stationary mean queue length and is determined using M/M/1/K formulae. It turns out from the analysis that the model do not work well in cases where the threshold probability of RED is small and the load is high. Also the model is numerically solvable only with *continuous* piecewise linear drop probability functions of RED.

Other works have focused on the behavior of closed-loop control protocols such as Transmission Control Protocol (TCP). TCP adjusts its sending rate according to the drop rate of packets

in the network ³. Thus, RED can be perceived as a controller of TCP sources via the generated drop probability.

Yeom and Reddy [130] considered a model in which TCP packets are marked according to a Reservation Window R . Two types of packets are generated by the TCP source, *in* and *out*. All packets in a TCP window are marked as *in* when the window size is less than R . If the window size exceeds R , the first R packets are marked as *in* and the rest as *out*. In the study they considered two particular situations, the under-subscription case and the over-subscription case. In the former, only *out* packets are dropped by the network. In the latter, all the *out* packets are dropped while *in* packets experience some losses. Such a marking scheme can be implemented using a token bucket with bucket size being zero. The distinction between the two cases supposes that the total number of reservation rates is specified and could not be variable.

Sahu *et al.* [110] carried out an analysis under similar assumptions, with however arbitrary bucket size for the token bucket. They connected the two situations to a particular configuration of RIO (Random Early Detection with In and Out) which is implemented in the router: the discard functions of *in* and *out* traffic do not overlap, in the sense that there is a cutting point of the average queue length below which only *out* packets are dropped (the under-subscription scenario), and above which all the *out* packets are dropped (the over-subscription scenario).

It is worthwhile noticing that the number of flows sharing a queue in a router is usually large and varies quickly, the queue length changes quickly as well. This variance increases significantly when RED is deployed in the router [92]. Thus, even if the discard functions do not overlap, a router can hardly stay in one of the scenarios during a time interval of length of the order of the TCP time scale. Thus, in all such time intervals, both *in* and *out* packets could experience loss simultaneously.

2.7 Conclusion

In this chapter we presented an overview of the different mechanisms of Differentiated Services. We then summarized some of the analytical studies that have been done to evaluate the performance of this framework. Our goal in the next two chapters is to develop models that extend the works of [91], [110] and [20] to study the interaction between TCP and RIO routers. We will use similar modeling techniques and also simulation to measure the error introduced by the simplifying assumptions used for mathematical tractability.

³A brief overview of TCP is provided in Section 3.1 of Chapter 3

Chapter 3

TCP Throughput in Networks with Differentiated Services

In this chapter we analyze the performance behavior of TCP in presence of multiple priority classes. We consider a bulky TCP connection that sends packets through a marker including one or more token buckets. Packets are assigned to a class according to the TCP transmission rate. The differentiation between classes is realized by attributing a drop probability to each class. The higher the priority of the class, the lower the drop probability. Our objective is to develop a general model that gives the expressions of the average window size and the average throughput of the TCP connection as a function of the token bucket parameters, the average round-trip time and drop probabilities. Our results are based on the deterministic analysis of the steady-state window size. We start by presenting a brief overview of TCP in the next section. Then, we present the framework and the general approach of our work. In Section 3.3, we focus on the case of two classes (“in” and “out” packets) and provide detailed analysis of the TCP window size for these two classes of traffic. In Section 3.4, we provide validation results based on the simulations using NS.

3.1 TCP in a Nutshell

TCP is a connection-oriented, end-to-end reliable protocol designed to achieve end-to-end flow control, error control and congestion control. The TCP receiver sends an acknowledgment packet (ACK) for each packet received by the TCP sender allowing for packet loss detection. When a packet is sent, a retransmission timer is initialized. Upon a receipt of an ACK, the timer is re-initialized. If no ACK is received before the expiration of the timer, a timeout event is triggered. The TCP sender is responsible for retransmitting the lost packets.

The transmission rate of TCP is controlled by the evolution of the TCP congestion window and the TCP advertised window. The former is in its turn controlled by congestion control mechanisms. The TCP sender is allowed to send up to the minimum between the advertised window and the congestion window. If the advertised window is large enough then the congestion

window represents the number of packets in transit between the sender and the receiver. In the following we will describe briefly the congestion control mechanisms of Reno version of TCP. A complete description is available in [104, 116, 71, 10].

3.1.1 Slow Start

The objective behind Slow Start is to avoid sending a large burst of packets that is beyond what the network can support at the beginning of the connection or after an idle period. It starts by sending one packet and then waiting for its ACK. When that ACK is received, the congestion window is doubled. Thus, two new packets can be sent. Afterwards, during the Slow Start period, for each ACK received the congestion window is incremented by one. This provides an exponential growth of the congestion window since each Round Trip Time (RTT) the window is doubled. This phase ends when a router in the path starts dropping packets or when the Slow Start threshold (*ssthresh*) is reached.

3.1.2 Congestion Avoidance

The Congestion Avoidance phase follows immediately the Slow Start phase. The congestion window is then set to *ssthresh*. When there is no congestion, the congestion window, called commonly "*cwnd*", is increased linearly following the additive increase principle [72]. Each time an ACK is received, the congestion window is incremented by $1/cwnd$, which means that each RTT the congestion window is incremented by one. When congestion occurs TCP must slow down its transmission rate. *ssthresh* is set to the half of the congestion window size. Hence, the next Congestion Avoidance period will start on half the previous congestion window size. This is the multiplicative decrease policy suggested in [72]. TCP assumes that the probability of packet loss caused by damage is very small, therefore the loss of packets notifies congestion.

3.1.3 Fast Retransmit

Fast Retransmit algorithm considers two indications of packet loss: the receipt of 3 duplicate ACKs or a timeout occurring. When a packet is lost and if the TCP sender receives the third duplicate ACK before the timer expires, it is assumed to be a strong indication that the packet has been lost. TCP then retransmits the missing packet.

3.1.4 Fast Recovery

After Fast Retransmit sends the missing packet Fast Recovery is called. If the packet loss was notified by a timeout event then TCP enter in the Slow Start phase. Otherwise if the packet loss was notified by a triple duplicate ACK, Slow Start is not performed. Fast Recovery starts when the missing packet is retransmitted and ends when its ACK is received by the TCP sender. During this period, the congestion window is inflated ([116]) to take into account the packets

that have left the network and for which the receiver has send a duplicate ACK. Fast Recovery fails when 3 or more packets are lost in a row from a window [41] and Slow Start is initiated.

3.2 TCP Throughput Analysis

There exist several studies on the throughput characterization of TCP. Most of them are concerned with the best-effort traffic. [85] and [89] establish a simple expression of the throughput of a bulky TCP connection (which always has something to send) as a function of the Round-Trip-Time (RTT) and the loss probability of packets p experienced by the TCP session: $thrpt = \frac{1}{RTT} \sqrt{\frac{3}{2p}}$. A more elaborated analysis of the throughput was provided in [101] which refines the above first-order approximation of the TCP throughput. Further studies are carried out in [8] on the mathematical characterization of the dynamics of TCP and in [7] on the non-Bernoulli type packet losses. In this section, we present a simple model to determine the expression of the throughput and some basic relations, some of which exist already in the literature.

3.2.1 Network Assumptions

We shall consider, as in most previous studies, a bulky TCP connection where the source has always data to send. We also assume that, when the source starts sending packets, resources are already allocated and different parameters are distributed (for example by means of bandwidth brokers). We consider only the Congestion Avoidance and Fast Retransmit algorithms to model TCP throughput. We suppose that for a bulky TCP connection, the contribution of Slow Start and Fast Recovery to the throughput is negligible. However, we consider all the features of TCP in the simulations we run to validate the model.

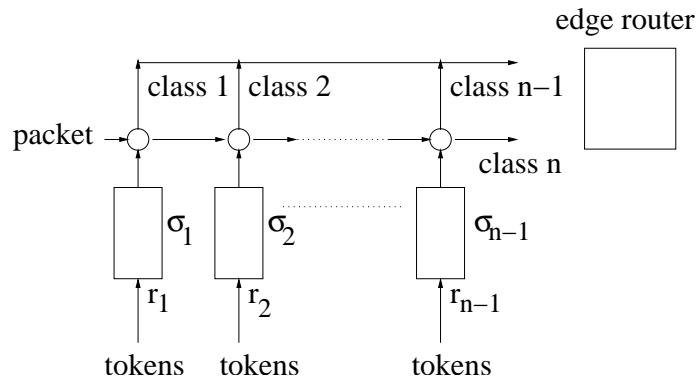


Figure 3.1: Packet marking using Token Buckets

We first consider the general case where TCP packets are marked using one or more token buckets depending on the number of classes of traffic in the network. Let n denote the total

number of classes, labeled $1, 2, \dots, n$, with decreasing preferences of treatment. The only assumption we make about the network outcome is that a dropping probability is measurable for any class i of a given TCP connection. These probabilities correspond to the average dropping probabilities at the “bottleneck” router of that connection. They could be the result of deploying RED in the routers or any other differentiation mechanism. These loss probabilities will be used as input parameters for the computation of the TCP throughput. Let p_1, p_2, \dots, p_n denote those probabilities. Obviously, we have $p_1 < p_2 < \dots < p_n$. The token buckets are defined by the rates of token generation and the bucket sizes denoted by (r_i, σ_i) , $i = 1, \dots, n - 1$ (Figure 3.1). Note that, here, for simplicity of analysis, we assume that the rates r_i are expressed in terms of packets instead of Bytes.

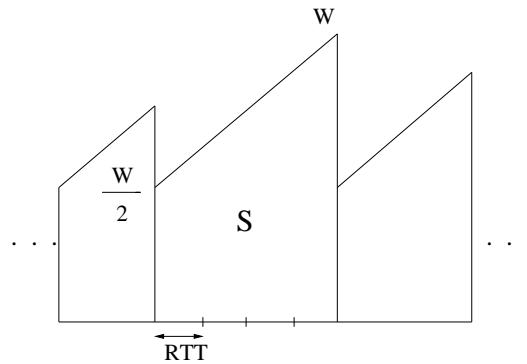


Figure 3.2: Window size evolution in congestion avoidance periods

3.2.2 Relationship between Throughput and Window Size of TCP

Although the analysis of the TCP Reno behavior provided in [101] was for best-effort traffic, some basic relations are still valid for our work on TCP with DiffServ. We recall some of these results here.

Consider the TCP window size evolution (see figure 3.2) in its steady state. Denote by

W : the window size of the TCP Reno protocol at the end of a Congestion Avoidance (CA) period;

S : the number of packets sent in the CA period;

T : the duration of the CA period;

R : the number of packets sent to retransmit a lost packet detected by a Timeout;

Z : the duration of the timeout retransmission period;

p_{TO} : the probability that a loss is detected by a Timeout.

Then, it follows from [101] that the throughput

$$B = \frac{S + p_{TO}R}{T + p_{TO}Z} \quad (3.1)$$

It is easy to see that $S = \frac{3W^2}{8}$ and $T = \frac{W}{2}RTT$ so that when there are few packet timeouts, the throughput and the window size relation is reduced to $B = \frac{S}{T} = \frac{3W}{4RTT}$

In general, a good approximation of timeout probability is given by (see [101]) $p_{TO} \simeq \frac{3}{W}$. The quantities R and Z are given by:

$$R = \frac{1}{1-p}, \quad Z = RTO \frac{1+p+2p^2+4p^3+8p^4+16p^5+32p^6}{1-p}$$

where RTO is the retransmit timeout constant (Actually, RTO is estimated dynamically by the source using RTT estimation, see [2] for a recent study), and p is the probability of loss. After a timeout, the transmission rate is very slow, consequently, buckets would be filled by tokens and then $p = p_1$

It is now clear that the throughput of a TCP connection is fully determined by the characterization of the steady state window size W .

3.2.3 Relation between Congestion Avoidance Area and Loss Probabilities

Let there be S packets in a CA period, among which S_i packets are of class i such that $S = \sum_{i=1}^n S_i$. Let y_j be the indicator function of the event that the j^{th} packet in a CA period is lost. Then, by definition of the CA period, $\sum_{j=1}^S y_j = 1$. Thus, by taking the expectation, we obtain the following loss model equation which is valid in the case of multiple classes of packets.

$$\sum_{i=1}^n S_i p_i = 1. \tag{3.2}$$

It is worthwhile noticing that in our analysis, we use deterministic (and heuristic) arguments, so that S_i 's are treated as constants. A rigorous analysis can be performed where S_i 's are random variables. In such a case, Equation (3.2) becomes $\sum_{i=1}^n E[S_i]p_i = 1$ and can be proved using stopping time theory and Wald's identity [74].

Observe that if $p_i = p$ for all $i = 1, 2, \dots, n$, i.e. no differentiation between packets, we find the equation established in the literature for the best-effort traffic: $Sp = 1$ which yields $W = \sqrt{\frac{8}{3p}}$.

3.3 TCP Window Size

It is clear from the previous section that the TCP throughput is fully determined by the CA window size. We now investigate how this window size is determined by the parameters of the token buckets. We shall only consider the case of $n = 2$ classes.

Let these classes be *in* packets and *out* packets, thus the source experiences two loss probabilities p_{in} and p_{out} . One token bucket is used for the marker with parameters (r, σ) .

We conduct the analysis in two steps. First we find the values of S_{in} and S_{out} using the CA period representation. Second, we solve equation (3.2) to determine the steady state congestion window W .

Define $r = \frac{\rho}{RTT}$. We use the same paradigm as that of [101] and [110] for our analysis. When the steady state is reached, the evolution of the window size becomes somewhat periodic and stochastically identical. Each case in figure 3.3 shows this evolution just for one period. We consider a discrete-time model, with the time slot being RTT. A burst of packets (the window size) are sent each RTT. RTTs are not shown in the figures in order not to encumber them. Let w denote the current window size. Then, w packets are sent during RTT. At the same time, ρ tokens are generated and eventually consumed. Tokens in the bucket can be increased or decreased depending on whether w is less or greater than ρ .

3.3.1 Triple Duplicate Events

Consider first the case when losses are detected by Triple Duplicates, i.e. the window is only divided by 2 at the end of the CA period. We will investigate 4 possible scenarios of a CA period. In order to simplify the analyses, we assume that the token bucket is initially empty. The reader will see later that for the three first cases the token bucket becomes empty at the end of the CA period. Thus, the assumption of the empty token bucket will be valid for the beginning of the next CA period. For the fourth case, it turns out the bucket size σ plays no role in the computation of the window size. Thus, we can simply set $\sigma = 0$ in this case so that the assumption of the empty token bucket at the beginning of the CA period is still valid.

Case 1: $\rho \leq W/2$

In this scenario, see case 1 in figure 3.3, the window consumes all the ρ tokens generated at each RTT ($w \geq \rho$). From figure 3.3 we can calculate the total number of *in* and *out* packets.

$$S_{in} = \rho \frac{W}{2}, \quad S_{out} = 3 \frac{W^2}{8} - \frac{W\rho}{2}$$

The loss model equation (3.2) can be rewritten as:

$$\rho \frac{W}{2} p_{in} + \left(3 \frac{W^2}{8} - \frac{W\rho}{2} \right) p_{out} = 1 \quad (3.3)$$

Case 2: $W/2 < \rho < W$, $\sigma \geq \frac{(\rho - \frac{W}{2})^2}{2}$, $\rho + \sqrt{2\sigma} \leq W$

At the beginning of this period there are more generated tokens than sent packets ($w < \rho$) thus all packets are marked *in* and $\rho - w$ tokens are added to the token bucket every RTT. This phase ends when the congestion window reaches the value of ρ . Since σ is large enough, the total number of accumulated tokens is then equal to $(\rho - \frac{W}{2})^2/2$ which is determined directly from Figure 3.3. When $w > \rho$, incoming packets are still marked *in* until the bucket becomes empty.

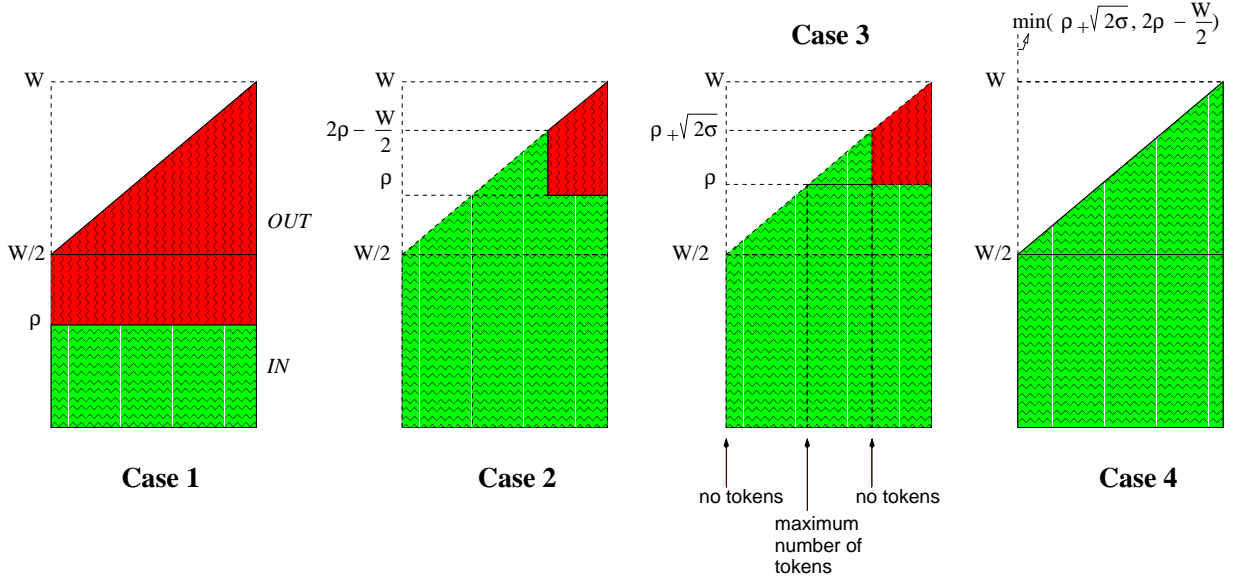


Figure 3.3: Packet marking

The first packet marked *out* is the packet number $(\rho + 1)$ sent at the $(2\rho - \frac{W}{2} + 1)^{th}$ *RTT*. Note that if there are enough accumulated tokens to make all the remaining packets in the period of type *in* then we fall into case 4.

Here again we can write:

$$\left(-\frac{1}{8}W^2 + \rho W - \frac{1}{2}\rho^2 + \frac{1}{2}\left(\rho - \frac{1}{2}W\right)^2\right)p_{in} + \left(\frac{1}{2}W^2 - \rho W + \frac{1}{2}\rho^2 - \frac{1}{2}\left(\rho - \frac{1}{2}W\right)^2\right)p_{out} = 1$$

After some simple computation, we obtain the same equation as (3.3).

Case 3: $W/2 < \rho < W$, $\sigma < \frac{(\rho - \frac{W}{2})^2}{2}$, $\rho + \sqrt{2\sigma} < W$

It is the same scenario as the previous one except that the bucket becomes full before the end of the accumulation phase (Figure 3.3, case 3).

Similarly, we write:

$$\left(-\frac{W^2}{8} + \rho W + \sigma - \frac{\rho^2}{2}\right)p_{in} + \left(\frac{W^2}{2} - \rho W - \sigma + \frac{\rho^2}{2}\right)p_{out} = 1 \quad (3.4)$$

Case 4: $(W \leq \rho)$ or $(\rho > \frac{W}{2}, W \leq \rho + \sqrt{2\sigma}, W \leq 2\rho - W/2)$

This scenario (Figure 3.3, case 4) is the case where all the packets in the period are *in*. The loss model equation (3.2) is reduced to the simpler form

$$3\frac{W^2}{8}p_{in} = 1 \quad (3.5)$$

By solving equations (3.3), (3.4) and (3.5) we obtain three expressions of W under different conditions. In the following we group and simplify these conditions.

Equation (3.3) holds both in case 2 and case 3 so when:

$$(\rho > W/2, 2\rho - W/2 < W \quad \text{and} \quad \sqrt{2\sigma} > \rho - W/2 \quad \text{and} \quad \rho < W) \quad \text{or} \quad \rho \leq W/2$$

After some manipulations of these conditions we get:

$$W \geq \max\left(\frac{4}{3}\rho, 2\rho - 2\sqrt{2\sigma}\right) \quad (3.6)$$

Equation (3.4) is obtained when

$$\rho > W/2 \quad \text{and} \quad \rho + \sqrt{2\sigma} < W \quad \text{and} \quad \sqrt{2\sigma} < \rho - W/2 \quad \text{and} \quad \rho < W$$

which yields

$$\rho + \sqrt{2\sigma} < W < 2\rho - 2\sqrt{2\sigma} \quad (3.7)$$

Equation (3.5) is obtained when

$$(\rho > W/2 \quad \text{and} \quad W \leq \min(\rho + \sqrt{2\sigma}, 2\rho - W/2)) \quad \text{or} \quad W \leq \rho$$

which gives:

$$W \leq \min\left(\frac{4}{3}\rho, \rho + \sqrt{2\sigma}\right) \quad (3.8)$$

We define $\delta = p_{out} - p_{in}$. After solving equations (3.3), (3.4) and (3.5), using the fact that $\delta > 0$ we get respectively three expressions of W :

$$W = \begin{cases} \sqrt{\frac{8}{3p_{in}}} & \text{(A) if } W \leq \min\left(\frac{4}{3}\rho, \rho + \sqrt{2\sigma}\right) \\ \frac{4\rho\delta + 2\sqrt{4\rho^2\delta^2 + 2(4p_{out} - p_{in})\left[\left(\sigma - \frac{\rho^2}{2}\right)\delta + 1\right]}}{4p_{out} - p_{in}} & \text{(B) if } \rho + \sqrt{2\sigma} < W < 2\rho - 2\sqrt{2\sigma} \\ \frac{\rho\delta + \sqrt{\rho^2\delta^2 + 6p_{out}}}{\frac{3}{2}p_{out}} & \text{(C) if } W \geq \max\left(\frac{4}{3}\rho, 2\rho - 2\sqrt{2\sigma}\right) \end{cases} \quad (3.9)$$

Notice that depending on whether ρ is less or greater than $3\sqrt{2\sigma}$ we can remove the max and min operators so as we get two simple cases that clearly indicate that we covered all possibilities.

It is simple now to substitute W in the conditions for each case. Consider first the case $\rho > 3\sqrt{2\sigma}$. Then, (3.9) can be rewritten as the following:

$$W = \begin{cases} \text{(C) if } \rho \leq \rho_1 \\ \text{(B) if } \rho_1 < \rho < \rho_2 \\ \text{(A) if } \rho \geq \rho_2 \end{cases} \quad (3.10)$$

Consider now the case $\rho \leq 3\sqrt{2\sigma}$. Then,

$$W = \begin{cases} \text{(C)} & \text{if } \rho \leq \rho_3 \\ \text{(A)} & \text{if } \rho > \rho_3 \end{cases} \quad (3.11)$$

where

$$\rho_1 = \frac{\sqrt{2\sigma}(2p_{out}+p_{in})+\sqrt{2\sigma\delta^2+2(2p_{in}+p_{out})}}{2p_{in}+p_{out}}, \quad \rho_2 = \sqrt{\frac{8}{3p_{in}}} - \sqrt{2\sigma} \quad \text{and} \quad \rho_3 = \sqrt{\frac{3}{2p_{in}}}$$

Here again we remark that if $\sigma = \frac{1}{12p_{in}}$ then $\rho_1 = \rho_2 = \rho_3 = 3\sqrt{2\sigma}$ so by combining equations (3.10) and (3.11) we finally obtain:

$$W = \begin{cases} \frac{\rho\delta + \sqrt{\rho^2\delta^2 + 6p_{out}}}{\frac{3}{2}p_{out}} & \text{if } \rho \leq \min(\rho_1, \rho_3) \\ \frac{4\rho\delta + 2\sqrt{4\rho^2\delta^2 + 2(4p_{out} - p_{in})\left[\left(\sigma - \frac{\rho^2}{2}\right)\delta + 1\right]}}{4p_{out} - p_{in}} & \text{if } \rho_1 < \rho < \rho_2 \\ \sqrt{\frac{8}{3p_{in}}} & \text{if } \rho \geq \max(\rho_2, \rho_3) \end{cases} \quad (3.12)$$

With these last two results (3.9) and (3.12) we can easily analyze the effects of the token bucket parameters and of the loss probabilities on the Window size of TCP. Figure 3.4 and 3.5 plots W as a function of token bucket parameters (ρ, σ) . The loss probability of *in* packets is fixed to 0.001 and we vary the loss probability of *out* packets. We choose low values since the current model is considered to be valid for small number of losses.

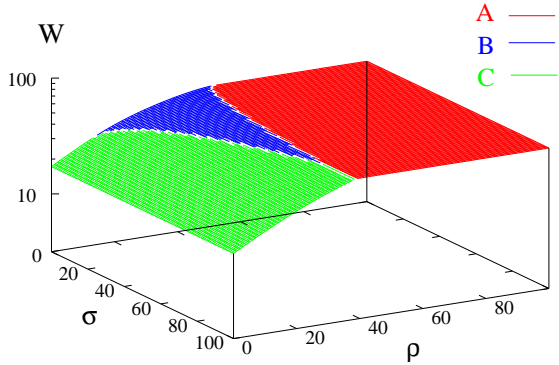


Figure 3.4: Window size function with $p_{in} = .001, p_{out} = .009$

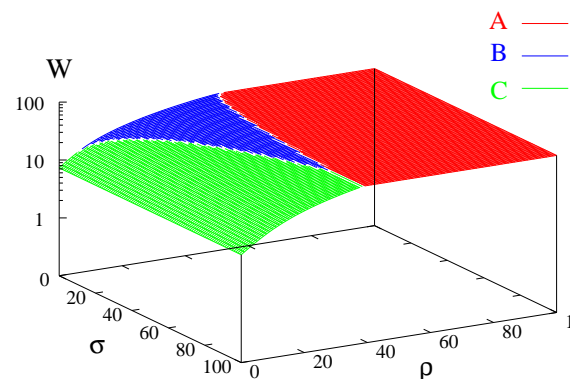


Figure 3.5: Window size function with $p_{in} = .001, p_{out} = .05$

We notice that there is no significant effect of the buffer size σ , whatever the value of ρ even if $p_{in} \ll p_{out}$. Also from the formula we remark that the maximum value of the bucket size that can affect the window size is $1/(12p_{in})$ since the terms (A), (C) and ρ_3 do not depend on σ . For example if the loss probability of *in* packets is 0.01 then it is not useful to allocate more than 8 tokens to the buffer of the token bucket.

An other observation is that the impact of σ increases with the drop probabilities. When $p_{out} = 0.009$ (Figure 3.4) and $\rho = 20$, an increase of 600% in σ results in an increase of 1% of the window. However, When $p_{out} = 0.05$ (Figure 3.5) and $\rho = 20$, an increase of 600% in σ results in an increase of 3% of the window. Intuitively, when the network is more severe with low priority

packets, it is necessary to produce more high priority packets (using a larger bucket) to improve the total throughput.

3.3.2 Timeout Events

In order to use equation (3.1) which takes into account Timeout events, we have to find the expression of p_{TO} the probability that a loss is detected by a timeout and S the number of packets sent in the CA period. For the calculation of S we should pay attention to the effect of σ . Indeed, in the first period after the retransmission of the lost packet detected by a Timeout and the Slow Start phase, the token bucket will not be empty any more. Thus, for this particular period we have to find the corresponding W^σ . For this goal we can assume that the bucket becomes completely full (this assumption was also made in [110]). This assumption is justified by the fact that few packets are sent after the window collapse. Using similar representations as in the previous section, we get three expressions of W^σ :

$$W^\sigma = \begin{cases} \frac{\frac{\rho}{2}\delta + \sqrt{\frac{\rho^2}{4}\delta^2 + \frac{3}{2}p_{out}[\sigma\delta + 1]}}{\frac{3}{4}p_{out}} & \text{if } \rho \leq \min(\rho_1^\sigma, \rho_3^\sigma) \\ \frac{4\rho\delta + 2\sqrt{4\rho^2\delta^2 + 2(4p_{out} - p_{in})\left[(\sigma - \frac{\rho^2}{2})\delta + 1\right]}}{4p_{out} - p_{in}} & \text{if } \rho_1^\sigma < \rho < \rho_2 \\ \sqrt{\frac{8}{3p_{in}}} & \text{if } \rho \geq \mathbf{1}(\sigma < \frac{1}{3p_{in}})\rho_2 + \mathbf{1}(\sigma \geq \frac{1}{3p_{in}})\rho_3^\sigma \end{cases} \quad (3.13)$$

where

$$\rho_1^\sigma = \frac{\sqrt{2\sigma\delta + 2}}{\sqrt{p_{out} + 2p_{in}}} \quad \text{and} \quad \rho_3^\sigma = \sqrt{\frac{3}{2p_{in}}}(1 - p_{in}\sigma)$$

Equation (3.1) can then be written as

$$B = \frac{(1 - p_{TO})\frac{3W^2}{8} + p_{TO}\frac{3W^\sigma}{8} + p_{TO}R}{(1 - p_{TO})\frac{W}{2}RTT + p_{TO}\frac{W^\sigma}{2}RTT + p_{TO}Z} \quad (3.14)$$

3.4 Throughput Model Validation

We now proceed with the simulations in order to validate the formula of TCP throughput, expressed as a function of four network parameters, the rate and the size of the token bucket used for the marking and the loss probabilities of *in* and *out* packets. For this purpose we used the NS-2 simulator [115] in which TCP congestion control algorithms are implemented. We also installed and fixed the existing patch that adds DiffServ functionalities [97] including RED with *in* and *out* (RIO). We used a single TCP source which keeps sending packets. Each simulation was run for one hour TCP connection time so as to reach the steady state.

The parameters of the token bucket that we specified in the simulator are (r, σ) . For each simulation, we measure the average RTT so we deduce ρ the number of incoming tokens per RTT, then we calculate the expected throughput given by our model. We count the total number of packets on the outgoing link of the source which gives the throughput of the simulation. Concerning RED configuration, we implemented an approximate scheme where each class of packets is associated with a dropping probability according to which packets are dropped independently of the queue length. Such an approximation was validated, again through NS-2, as follows. First, we set arbitrary values for RED and then we measure the loss probabilities and the throughput. Then, we use the same probabilities to drop packets randomly without considering the queue length. We found the same steady state throughput and also the same average RTT observed in the previous step. Therefore, we used this approximate scheme for the validation of the analytical results. The motivation behind this approximation is to allow us to vary the loss probabilities which is a difficult task if we only manipulate the RED parameters. Indeed, it is not obvious to assess the steady state loss probabilities due to the feedback control mechanism of TCP.

We begin by validating equation (3.12) in the two cases $\sigma > \frac{1}{12p_{in}}$ and $\sigma \leq \frac{1}{12p_{in}}$. Figures 3.6 and 3.7 correspond to these two cases. We observe that the shape of the analytical curve $thpt(\rho)$ is the same as the experimental throughput. This shows that the three conditions established in Section 3.3 are accurate and they cover all the possible scenarios.

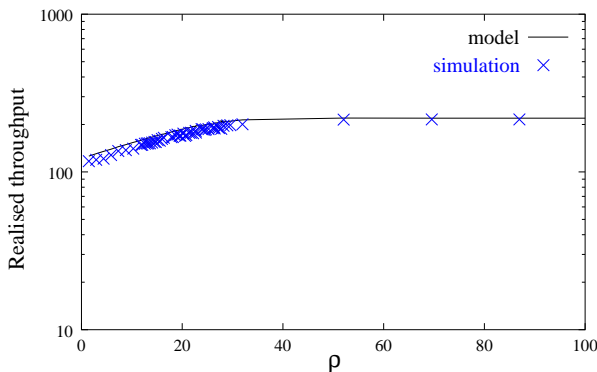


Figure 3.6: Throughput with $\sigma = 60$ packets, $p_{in} = .001$, $p_{out} = .005$

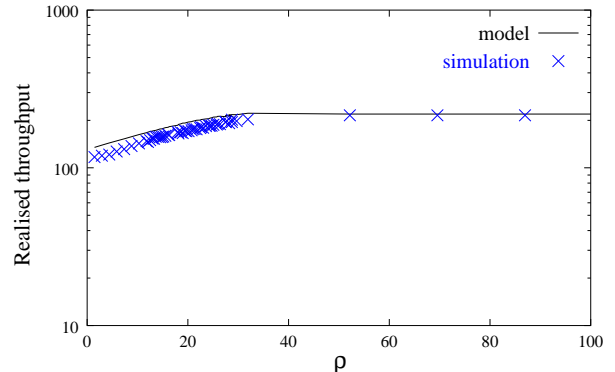


Figure 3.7: Throughput with $\sigma = 100$ packets, $p_{in} = .001$, $p_{out} = .005$

We can notice that the theoretical throughput is always greater than the simulation. The explanation is that our model considers a linear increase of the congestion avoidance window rather than a sub-linear one which is the normal behavior. The sub-linearity comes from the fact that when the window increases, the queueing delay increases causing an increase in the RTT . This approximation over-estimates the throughput [6]. The use of the average RTT in the model instead of the round-trip propagation delay reduces the engendered bias.

Now let us analyze the throughput as a function of the loss probabilities. The average RTT employed in our simulations is about 141ms. $RTO=1.2s$. Tables 3.1, 3.2, 3.3 and 3.4 illustrate

the comparison results.

Table 3.1: Measured data and theoretical predictions for the case of $r = 100$ packets, $\sigma = 20$ packets

p_{in}	p_{out}	Analytical throughput	Experimental throughput
0.001000	0.002000	193.492849	195.014603
0.001000	0.006000	149.899933	145.958992
0.002000	0.005000	143.746846	142.929386
0.002000	0.009000	127.126651	124.842969
0.003000	0.008000	121.263124	120.042809
0.004000	0.008000	112.613819	110.537708
0.010000	0.040000	70.661010	67.359072
0.020000	0.050000	42.986539	43.489498
0.040000	0.090000	23.567842	23.031606
0.100000	0.500000	8.517887	7.494299

Table 3.2: Measured data and theoretical predictions for the case of $r = 200$ packets, $\sigma = 80$ packets

p_{in}	p_{out}	Analytical throughput	Experimental throughput
0.000100	0.000500	479.514168	476.312062
0.000100	0.000900	396.754960	392.902581
0.001000	0.004000	236.721352	217.237047
0.001000	0.005000	232.851398	210.025805
0.001000	0.006000	230.006086	207.201440
0.002000	0.005000	186.426437	175.096019
0.003000	0.009000	149.037548	144.371274
0.010000	0.040000	70.223559	74.970794
0.020000	0.050000	40.180262	45.589518
0.090000	0.100000	5.713280	8.921384

We performed different sets of simulations, for each we fix the token bucket parameters (r, σ) and we vary the loss probabilities such that we get some throughput values greater than the token rate r which means that both *in* and *out* packets are generated. Again, we can observe that the prediction of the throughput of a TCP connection is accurate in all these situations.

Table 3.3: Measured data and theoretical predictions for the case of $r = 50$ packets, $\sigma = 20$ packets

p_{in}	p_{out}	Analytical throughput	Experimental throughput
0.002000	0.004000	136.814934	136.309662
0.002000	0.008000	104.805816	104.538908
0.004000	0.006000	107.007307	106.955791
0.004000	0.008000	96.743076	97.397479
0.010000	0.020000	59.894222	61.456891
0.010000	0.040000	51.838523	54.004401
0.020000	0.050000	39.681433	39.645729
0.020000	0.090000	38.153830	37.614923
0.060000	0.100000	15.557144	14.780956
0.100000	0.500000	8.517887	7.494299

Table 3.4: Measured data and theoretical predictions for the case of $r = 100$ packets, $\sigma = 80$ packets

p_{in}	p_{out}	Analytical throughput	Experimental throughput
0.001000	0.002000	221.776220	217.083617
0.002000	0.003000	174.009013	171.066013
0.002000	0.007000	154.410658	137.857171
0.003000	0.004000	146.062600	142.017804
0.003000	0.008000	140.278731	123.342869
0.004000	0.005000	126.348720	124.932386
0.004000	0.008000	126.347917	116.701540
0.010000	0.060000	70.224109	71.412082
0.020000	0.090000	40.180108	45.981996
0.200000	0.600000	1.137251	2.122825

3.5 Conclusion

In this chapter we have presented new analytical results on the throughput of TCP in networks with differentiated services. Our analyses have been focused on the case with token buckets as packet marking mechanism at the edge routers and an arbitrary packet dropping mechanism in the core routers. We have also provided simulation results in comparison to the analytical

ones. These results indicate that our formulas predict quite accurately the TCP-Reno behavior in DiffServ networks.

The closed form expressions derived in this chapter can be used in two applications. First, the network can control the transmission rate of real-time traffic to share fairly the total bandwidth, in other words, force real-time applications to be *TCP-friendly*. One way to do that is to measure periodically the drop probabilities of packets for each class in the network, then the edge router adjusts the rate of real-time traffic (by means of, for example, shaping) with respect to the rate computed by the expression of TCP throughput. The second application is network design and parameterization. First, we already quantified the impact of the buffer size (or burst size tolerance) of the packet marker on the average throughput. This impact is bounded, which means that beyond some threshold it is not useful to increase the tolerance.

Second, by analyzing closely the network we can determine the loss probabilities and the round-trip time as a function of network parameters such as the router capacity and the buffer size. Then, we can study the impact of network parameters on the behavior of TCP. The throughput of TCP could be used to tune network parameters for example in order to maximize this throughput, or to assign a fixed share of the total bandwidth to the TCP traffic traversing a DiffServ network. However, because of the *feedback* control mechanism of TCP, the analysis of the network must be conducted simultaneously with TCP. We will address this issue in the next chapter.

Chapter 4

Performance Analysis of TCP with RIO Routers

In this chapter we study the performance of Random Early Detection mechanism with two classes of service In-profile and Out-of-profile (RIO). We use the expressions of the throughput developed in the previous chapter and standard queueing analysis to describe a system of RIO queue fed by a set of TCP sources. RIO is a mechanism that includes both Active Queue Management for congestion control and preferential packet treatment for service differentiation. RIO is characterized by two non-decreasing drop probability functions. The most analyzed cases are piecewise linear functions defined by two thresholds and a maximum drop probability. When the average queue size is below the lower threshold, no packet is dropped. When the average queue size is in between the two thresholds, packets are dropped randomly according to the drop probability function. Beyond the upper threshold, any incoming packets are dropped. Each function is assigned to one class, i.e. *in* or *out*. The scheduling mechanism associated with RIO is FIFO which makes it suitable for multiplexing *in* and *out* packets of a TCP connection in the same queue because there is no reordering of packets.

As introduced in [52] and [33], the drop probability function uses a smoothed estimation of the average queue length to compute the drop probabilities. In this chapter we will instead consider the instantaneous queue length. Also, we will consider the total number of packets (instead of the total number of *in* packets) in the queue to compute the drop probability of *in* packets. However, May *et al.* [91] show by simulation that the impact of both schemes are comparable. In all cases, to give better service to *in* packets than *out* packets we need to set carefully the parameters of the drop probability functions.

Many previous studies have been carried out to characterize the steady state and the transient behavior of Active Queue Management in interaction with TCP traffic. Three approaches have been used to study this feedback system. The first approach uses differential equations to describe the *transient* behavior of TCP sources and queue *dynamics* [75, 79, 96, 63, 80, 24]. The equations of TCP can describe the evolution of the window size or the evolution of the packet rate. The equations that describe TCP are derived by computing the expectation of the change in the

arrival rate (window).

The second approach uses fixed-point approximations and it is first introduced to study a best-effort network in presence of TCP in [23] then a network with AQM routers in [49, 20]. The method consists of determining the average values of the queue length, the drop probability and the throughput of TCP using a set of equations that describe the TCP sources and the network in the *steady state* (when the TCP/RED system reaches an equilibrium point). The equations consider only the average measures and not the steady state distributions. The equations are solved numerically using a reciprocal tuning.

The third approach considers the interaction between TCP sources and RED as a Lagrangian method to solve an optimization problem [83, 25]. Each TCP source is considered to *implicitly* optimizing a utility function $U(x)$ where x is the transmission rate. The objective is to find the rate that maximizes the sum of these utility functions with the constraint that the total source rate does not exceed the capacity of the RED router. The Lagrangian method transforms this problem to an other problem without constraints called the dual problem. The new variable of the dual problem is perceived as the RED queue length. The utility function is derived from the congestion control algorithm implemented by TCP. This method is general and can be applied to many versions of TCP (Reno, Vegas) and AQMs. In this case, the variable of the dual problem will represent some congestion measure not necessarily the queue length.

In this chapter we will use the fixed point approach to analyzing the performance characteristics of TCP sessions in networks with RIO routers. We consider the case with a large number of TCP sessions which use token buckets for marking *in* and *out* packets at the entrance of the network. Under some simplifying assumptions we derive a set of equations that govern the evolution of these TCP sessions and the routers under consideration. We then solve these equations numerically using the fixed point method. We validate this model through simulations. We then use this method to study the impact of the RIO parameters on the performance characteristics of TCP sessions. The results of this work has also been published in [87, 88].

4.1 Network Model

We consider a RIO router in the network fed by N long-lived TCP connections. For simplicity of exposition, we shall assume that this router is the bottleneck router of these TCP sessions so that the round trip time (RTT) of these sessions are represented by the propagation delays and the queueing delay incurred in the router under consideration. However, as we shall see later on, the analysis techniques can be extended to the case of any arbitrary number of routers.

The router is modeled by a FIFO queue with RIO, see Figure 4.1. The RTTs are arbitrary and can be different for different TCP sessions. Every TCP session uses a token bucket (TB) to mark the packets *in* or *out*. The token bucket parameters are the rate generation of the tokens r^i and the buffer size of the bucket σ^i . Roughly speaking, if the instantaneous rate of the connection is less than r^i , the packets are marked *in*, otherwise they are marked *out*. The buffer

σ^i allows for burst absorption.

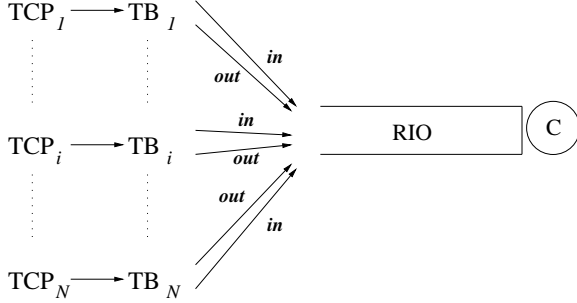


Figure 4.1: The simplified network model

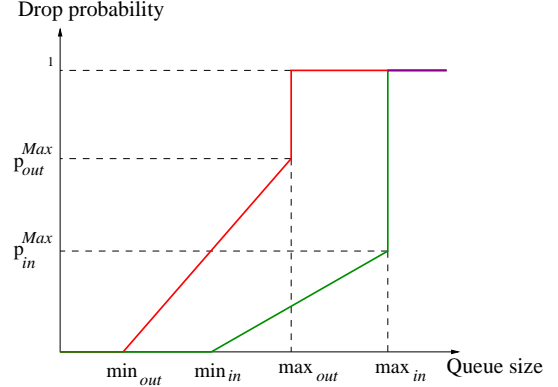


Figure 4.2: RIO drop probability functions

We now introduce the notation that will be used in the next section. The superscript i refers to the connection number and the subscript c refers to the class of packets (*in* or *out*).

- N : Total number of TCP connections
- T^i : Throughput of TCP connection i
- T : Total Throughput, i.e. $T = \sum_{i=1}^N T^i$
- T_c^i : Throughput of class c packets of TCP connection i
- T_c : Total throughput of class c packets
- (r^i, σ^i) : token bucket parameters of TCP connection i
- C : router capacity
- D^i : the round-trip propagation delay of connection i
- RTT^i : the round-trip time of TCP connection i
- RTO^i : the retransmission timeout of connection i
- \bar{q} : the average queue length in the router
- \bar{p}_c : average loss probability of class c packets
- $p_c(\cdot)$: drop probability function of class c packets (Figure 4.2), where

$$p_c(q) = \begin{cases} 0 & \text{if } q \leq \min_c \\ p_c^{\max} \frac{q - \min_c}{\max_c - \min_c} & \text{if } \min_c < q < \max_c \\ 1 & \text{if } q \geq \max_c \end{cases}$$

Where q is the instantaneous queue length.

4.2 Characteristic Equations and Computational Scheme

In this section we shall first derive a set of equations that relate the state variables of the TCP sessions and those of the router. We then use the fixed point method to numerically compute these performance metrics.

4.2.1 Equations of TCP Dynamics

Consider first the dynamics of the TCP control protocol. Using the same graphical analysis of the previous chapter, we can derive the expressions of the expected throughput of each class of packets (*in* and *out*) for each TCP connection as a function of the average loss probabilities, the token bucket parameters, the round-trip time and the retransmission timeout. For each connection i we have:

$$T_{in}^i = \frac{(1 - p_{TO}^i)S_{in}^i + p_{TO}S_{in}^{\sigma,i} + p_{TO}^i R^i}{(1 - p_{TO}^i)Y^i + p_{TO}^i Y^{\sigma,i} + p_{TO}^i Z^i} \quad (4.1)$$

$$T_{out}^i = \frac{(1 - p_{TO}^i)S_{out}^i + p_{TO}S_{out}^{\sigma,i}}{(1 - p_{TO}^i)Y^i + p_{TO}^i Y^{\sigma,i} + p_{TO}^i Z^i} \quad (4.2)$$

where

- S_c^i denotes the total number of class c packets sent in a congestion avoidance period following a triple duplicate loss, $c \in \{in, out\}$; Y^i is the duration of such a period;
- $S_c^{\sigma,i}$ is the total number of class c packets sent in a congestion avoidance period following a timeout loss event, $c \in \{in, out\}$; $Y^{\sigma,i}$ is the duration of such a period;
- p_{TO}^i the probability that a loss is detected by a Timeout;
- Z^i the duration of the timeout retransmission period;
- R^i the number of packets sent to retransmit a lost packet.

The term $p_{TO}^i R^i$ represents the number of packets sent in a timeout period. During this period we assume that the transmission rate is slow enough so that all packets are marked *in*.

The formulae of these expected parameters are presented at the end of this chapter in Appendix A.

We have also $T = \sum_{i=1}^N T^i$, $T_{in} = \sum_{i=1}^N T_{in}^i$ and $T_{out} = \sum_{i=1}^N T_{out}^i$.

The last equation is an approximation to compute the retransmission timeout:

$$RTO^i = RTT^i + \alpha^i \quad (4.3)$$

Actually, the instantaneous value of RTO is estimated with $RTO = SRTT + 4RTTVAR$ where $SRTT$ is a smoothed estimate of RTT and $RTTVAR$ is a smoothed estimate of the variation of

RTT. We observed in our simulations that this variation is negligible compared to the Round Trip Time. We observed also that the average RTT plus the preset lower bound is a good approximation of the average RTO. Since in many TCP implementations, e.g. BSD, the RTO is lower bounded by 1 second [2] we choose α^i equal to 1 second.

4.2.2 Equations of the RIO Router

For simplicity of analysis, we shall assume that the traffic entering the router is Poisson with rate equal to the sum of the throughputs of all TCP connections. We shall also assume that the service times in the router are exponentially distributed with parameter equal to the capacity of the router. The Poisson assumption seems to be justified when the TCP connection rate increases [22]. These assumptions allow us to derive analytically the expressions relating the average queue length and the loss probabilities. We remark in passing that under such assumptions we can also determine the departure process out of the queue so that we can resolve a system of multiple routers.

It is easy to see that the queue length is a birth-death process [91] so that it has the stationary distribution expressed as

$$\pi(i) = \pi(0) \left(\frac{T}{C}\right)^i \prod_{j=0}^{i-1} [1 - p(j)], \quad i = 1 \cdots \max_{in} \quad (4.4)$$

where

$$p(i) = \frac{T_{in} p_{in}(i) + T_{out} p_{out}(i)}{T_{in} + T_{out}}.$$

We assume, without loss of generality, that the minimal condition to give preferential service to *in* packets is $\max_{out} \leq \max_{in}$. Hence, $\pi(0)$ is given by the normalization equation

$$\pi(0) = \left[1 + \sum_{i=1}^{\max_{in}} \left(\frac{T}{C}\right)^i \prod_{j=0}^{i-1} [1 - p(j)] \right]^{-1}.$$

We shall also approximate the average loss probability of *in* packets and *out* packets observed for each connection \bar{p}_{in}^i and \bar{p}_{out}^i by the loss probabilities observed in the RIO queue \bar{p}_{in} and \bar{p}_{out} . However, the average loss probability observed by each connection including both *in* and *out* is different from the loss probability observed at the RIO queue.

The loss probabilities \bar{p}_{in} and \bar{p}_{out} are obtained using PASTA property of the Poisson process [91]:

$$\bar{p}_{in} = \sum_{i=0}^{\max_{in}} p_{in}(i) \pi(i) \quad (4.5)$$

$$\bar{p}_{out} = \sum_{i=0}^{\max_{in}} p_{out}(i) \pi(i) \quad (4.6)$$

The last equation determines the round-trip time by summing the total propagation delay to the average queueing delay.

$$RTT^i = D^i + \frac{\bar{q} + 1}{C} \quad (4.7)$$

4.2.3 Fixed Point Method

The above equations are now solved using a fixed point method. As depicted in Figure 4.3 Each iteration of the algorithm contains only two steps. In the first step we use the values of the throughput to determine a new load of the system. This load determines a new stationary distribution of the queue length and new values of the loss probabilities. In the second step we use the formulae of TCP throughput to update the throughput of one connection at a time which leads to a small update of the total throughput. All connections contribute on the total throughput after N iterations. This method is necessary in order to avoid a significant increase/decrease of the throughput at each iteration which could result in oscillations.

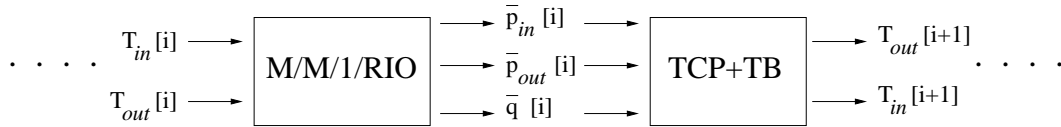


Figure 4.3: Iteration Scheme

4.3 Model Validation

In this section we validate our model using the NS-2 simulator. We simulate 100 TCP connections with 1 Mb/s access link for each connection. The bottleneck link capacity is set to 48 Mb/s . The Round-Trip propagation delays are chosen between 100 ms and 300 ms such that the Round-Trip propagation delay = $100 + 2i$ $i \in 0 \dots N - 1$. r^i and σ^i are fixed to 40 packets/sec and 20 packets . The packet size is equal to 1000 bytes. We run simulations for 3 scenarios of the RIO configuration corresponding to whether the drop probability functions fully overlap, partially overlap or do not overlap.

4.3.1 Fully Overlapped Case

In this case $\min_{in} = \min_{out} = 20$ and $\max_{in} = \max_{out} = 100$. We choose 0.01 and 0.06 for p_{in}^{max} and p_{out}^{max} , respectively. Table 4.1 compares the results obtained by a 30-minutes simulation with the numerical results obtained by the model. We observe that our model predicts the average parameters very accurately. Figure 4.4(a) plots the theoretical and the simulation throughput of all TCP connections. As in the previous chapter, we observe again that in most cases the model over estimates the throughput.

Table 4.1: The fully overlapped case

	Analytical	Simulation
T (pkts/sec)	5927.97	5810.57
T_{in} (pkts/sec)	3764.09	3707.57
\bar{p}_{in}	0.004134	0.003260
\bar{p}_{out}	0.015095	0.015937
\bar{q} (pkts)	33.79	31.18

4.3.2 Non-Overlapped Case

In this scenario, $max_{out} \leq min_{in}$. We drop all the *out* packets before start dropping *in* packets. The RIO parameters are the following: $min_{out} = 20$, $max_{out} = 60$, $p_{out}^{max} = 0.06$, $min_{in} = 60$, $max_{in} = 100$ and $p_{in}^{max} = 0.01$. The results of this case are reported in Table 4.2. We observe that the error for the throughput in this case is higher than the previous case, (Figure 4.4). However, this is counterbalanced by a lower error for the loss probabilities.

Table 4.2: The non-overlapped case

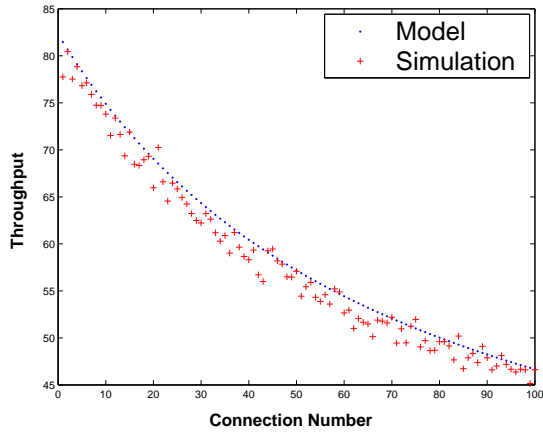
	Analytical	Simulation
T (pkts/sec)	5854.04	5648.91
T_{in} (pkts/sec)	3757.99	3617.91
\bar{p}_{in}	0.000005	0.000001
\bar{p}_{out}	0.023605	0.023314
\bar{q} (pkts)	21.70	17.63

4.3.3 Partially Overlapped Case

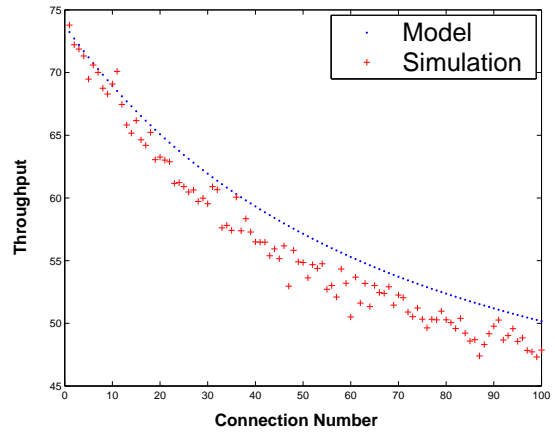
The drop probability functions overlap partially when $min_{in} < max_{out}$. We choose the following parameters: $min_{out} = 20$, $max_{out} = 100$, $p_{out}^{max} = 0.06$, $min_{in} = 60$, $max_{in} = 100$ and $p_{in}^{max} = 0.01$. We choose $max_{in} = max_{out}$ which is a similar configuration to the one set by default in CISCO routers. Again for this case the simulation results are close to the numerical results (Table 4.3, Figure 4.4(c)).

Table 4.3: The partially overlapped case

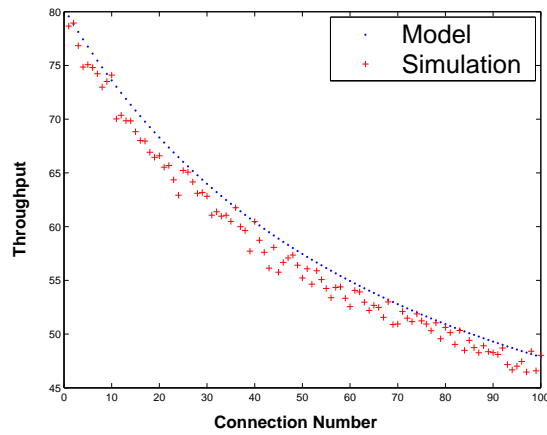
	Analytical	Simulation
T (pkts/sec)	5936.85	5798.03
T_{in} (pkts/sec)	3766.17	3711.03
\bar{p}_{in}	0.003145	0.002477
\bar{p}_{out}	0.016698	0.017428
\bar{q} (pkts)	35.65	32.69



(a) Fully overlapped case



(b) Non-overlapped case



(c) Partially Overlapped case

Figure 4.4: Throughput of TCP connections with different RTTs

4.4 Impact of RIO Parameters on QoS and Fairness

In this section, we use our analytical model to study the ways to set the RIO parameters and the effect of these parameters on the QoS and fairness of TCP sessions. The performance metrics under consideration are essentially the achieved throughput, the loss probability and the network delays. These quantities could be the key elements in a Service Level Agreement (SLA). Following the model in section 4.1, there are eight parameters that could have effect on these performance metrics. In this section, we will focus on the RIO parameters. In the following numerical examples, we consider mainly the same network configuration used for simulation in the previous section.

4.4.1 Throughput

In this DiffServ framework, each TCP connection attempts to achieve its reservation rate. In other words the constraint is $throughput^i \geq r_{SLA}^i$. In the experiment presented in section 4.3.1, all the connections have throughputs above the reservation rate. The throughput of TCP connection is inversely proportional to the RTT. A TCP connection with a large RTT may not be able to achieve its target rate. To illustrate that we add 5 connections to the set of 100 connections. We assign the following RTTs 0.4s, 0.5s, 0.6s, 0.7s and 0.8s. Figure 4.5 shows that neither of the connections could achieve the reservation rate. This is due to the fact that connections with small RTTs are very aggressive and they send many *out* packets beyond the reservation.

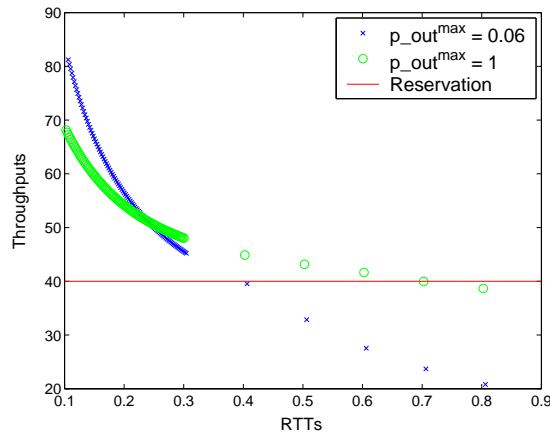


Figure 4.5: Throughput of TCP connections with variable RTTs

In order to reduce this unfairness, one solution is to increase the drop probability of *out* packets. Thereby, causing a decrease in the throughput of large RTT TCP connections and hence an increase in the throughput of small RTT connections. Figure 4.5 shows that by setting p_{out}^{max} to 1, 4 connections achieve the reservation. Figure 4.6 shows the evolution of the throughput vs. p_{out}^{max} . Generally, if we set RIO parameters such that $\bar{p}_{out} = 1$ and if there are still connections which cannot achieve the reservation, then either the capacity of the link or the buffer thresholds

should be re-provisioned.

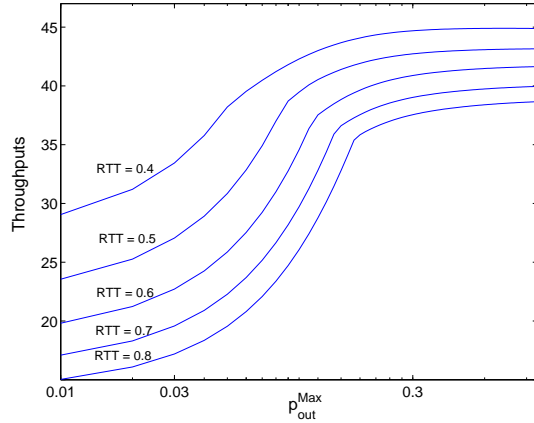


Figure 4.6: Impact of p_{out}^{max} on the TCP throughput

4.4.2 Delay

We can transform the constraints on the delay and the average delay to the queue length and the average queue length. To satisfy the strict constraint $delay \leq delay_{SLA}$, one simple way is to set $max_{in} = delay_{SLA} * C$. Then we can focus on the other parameters to satisfy other constraints.

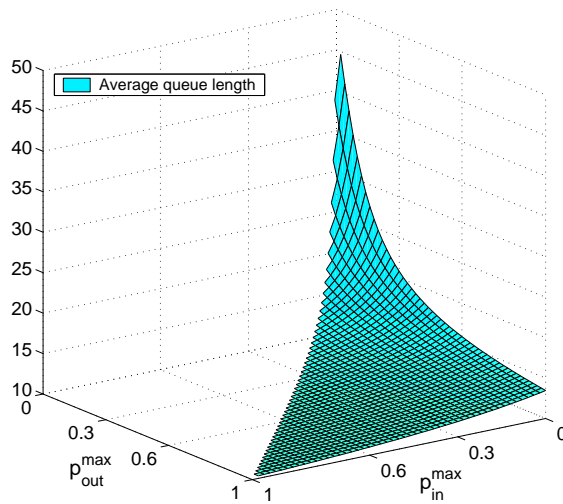


Figure 4.7: Effect of the drop probability thresholds on the average queue length

Figure 4.7 illustrates the average length vs. p_{in}^{max} and p_{out}^{max} . We can notice that when $p_{in}^{max} = 0$, i.e. when TD mechanism is deployed, the average is higher. Also that the p_{out}^{max} controls more the queue utilization. If p_{out}^{max} decreases, we increase significantly the utilization. This is due to the fact that we allow *out* packets to fill the buffer of the router. This is opposite

to the fairness goal mentioned in the previous paragraph since for that goal we need to increase the p_{out}^{max} . To study more the impact of \bar{q} , we keep p_{in}^{max} and p_{out}^{max} fixed and we vary min_{in} and min_{out} to cover all the cases where the drop functions partially overlapped. We do that for two values of p_{out}^{max} . We see from Figure 4.8 that the utilization does not rely much on min_{in} , though the performance is a little better when $min_{in} = max_{in}$.

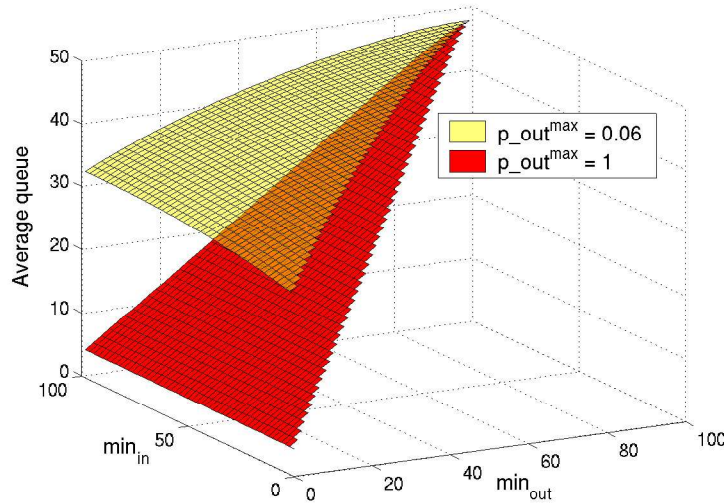


Figure 4.8: Effect of the minimum thresholds on the average queue length

4.4.3 Drop probability

We first consider again the fully-overlapped case of section 4.3.1. Suppose that in the SLA we want to ensure that the loss probability of *in* packets \bar{p}_{in} is less than p_{SLA} . Figure 4.9 shows the possible values of the two drop probability thresholds of RIO (p_{in}^{max} , p_{out}^{max}) that achieve the $p_{SLA} = 0.006$. Note that the condition $p_{in}^{max} = 0$ is not sufficient to ensure the desired p_{SLA} . In this scenario, we need to set p_{out}^{max} to at least 2.49%.

More generally, we can determine the upper and lower bounds of $\bar{p}_{in}(\cdot)$. Figure 4.12 plots the average loss probability of *in* packets as function of the drop probability thresholds p_{in}^{max} and p_{out}^{max} . The lower and upper bounds are mentioned in the Figure. If p_{SLA} is greater than the upper bound, we can satisfy the constraint $\bar{p}_{in} \leq p_{SLA}$ and thus achieve the SLA for all $(p_{in}^{max}, p_{out}^{max})$ settings. In this case, we can set $(p_{in}^{max}, p_{out}^{max})$ to control the level of differentiation between *in* packets and *out* packets. If p_{SLA} is less than the lower bound we can not achieve the SLA. However, if p_{SLA} is between the two bounds, we can achieve only if we set correctly the parameters $(p_{in}^{max}, p_{out}^{max})$. Figure 4.11 illustrates many examples. We notice that in all these cases the operating point should be close to the line which delimits the feasible region in order to increase the utilization of the network, i.e., we should choose p_{out}^{max} as low as possible and $p_{in}^{max} = 0$ (which means that the TD is in place). Similar observations can be made from Figure 4.10 which corresponds to different partially overlapped cases ($min_{out} \leq min_{in}$)

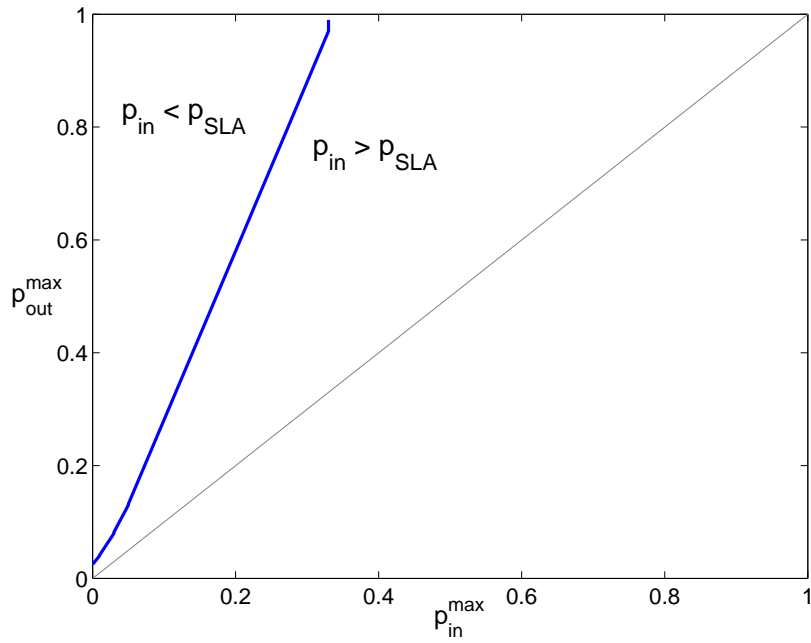


Figure 4.9: Feasible region for the loss probability constraint

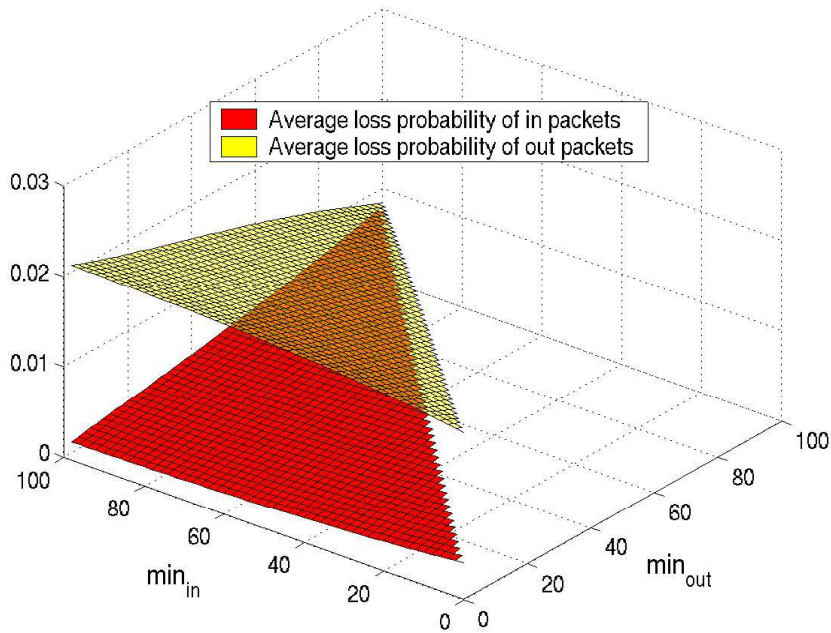


Figure 4.10: Effect of the minimum thresholds on the average loss probabilities

An important fact is that decreasing p_{in}^{max} does not contradict the rate goals, i.e., for all the QoS constraints, a very small value of p_{in}^{max} yields good performance. Also we notice that in the partially overlapped case, we obtain better performance in terms of loss probability and utilization when $min_{in} = max_{in}$ which corresponds to TD too.

In contrast, p_{out}^{max} has significant and *different* effects on the overall QoS. For example, we can configure differently the AF classes. If we know that in one AF class we will have TCP connections of almost the same RTTs we choose a low value of p_{out}^{max} . In an other AF class we can aggregate heterogeneous TCP connections that have tighter constraints on the throughput.

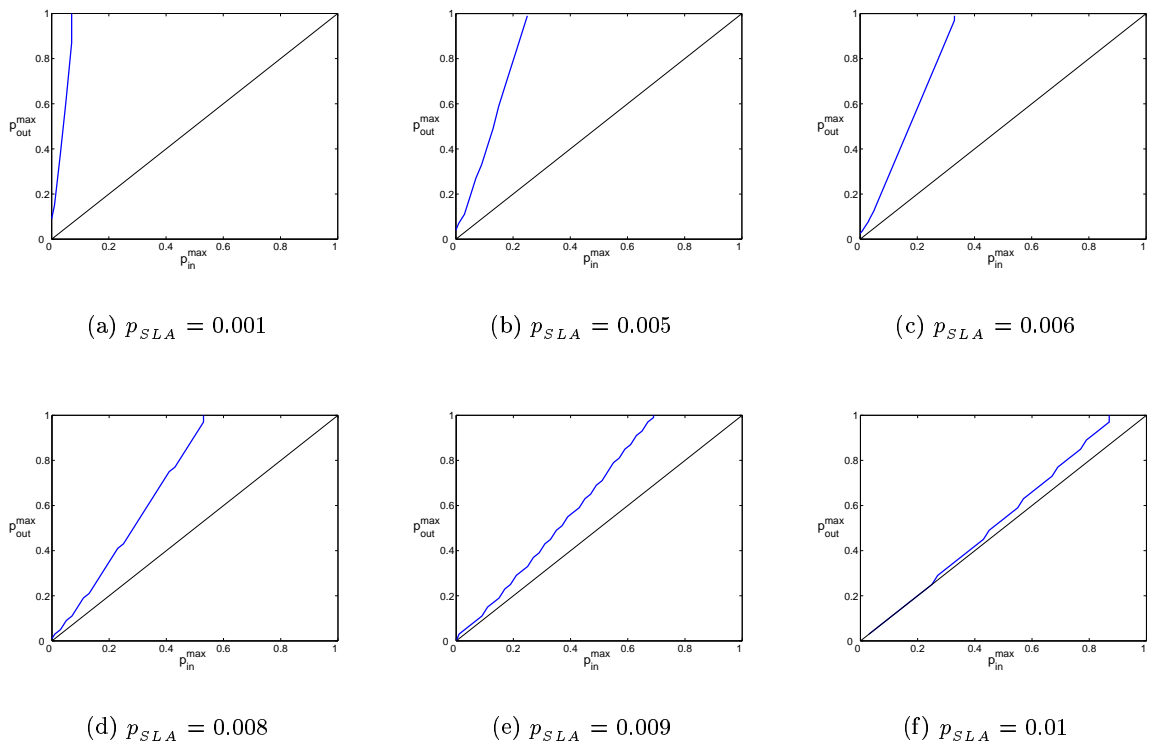


Figure 4.11: Feasible region for some loss probability constraints ($p \leq p_{SLA}$)

4.5 Conclusion

In this chapter we developed and validated a method to analyze the steady state behavior of long lived TCP connections in interaction with RIO routers. Using the model, we examined the impact of RIO thresholds on the QoS and fairness of TCP connections. The model can also be used to derive guidelines for the design of an AF class. We have shown that the drop probability threshold of *out* packets has a significant effect on the TCP throughput and on the average queue length. Setting this parameter consists in trading off between the network utilization and the fairness among TCP connections. For example, we can set the drop probability threshold to a small value when aggregating TCP connections with the same Round-Trip Time. We have also

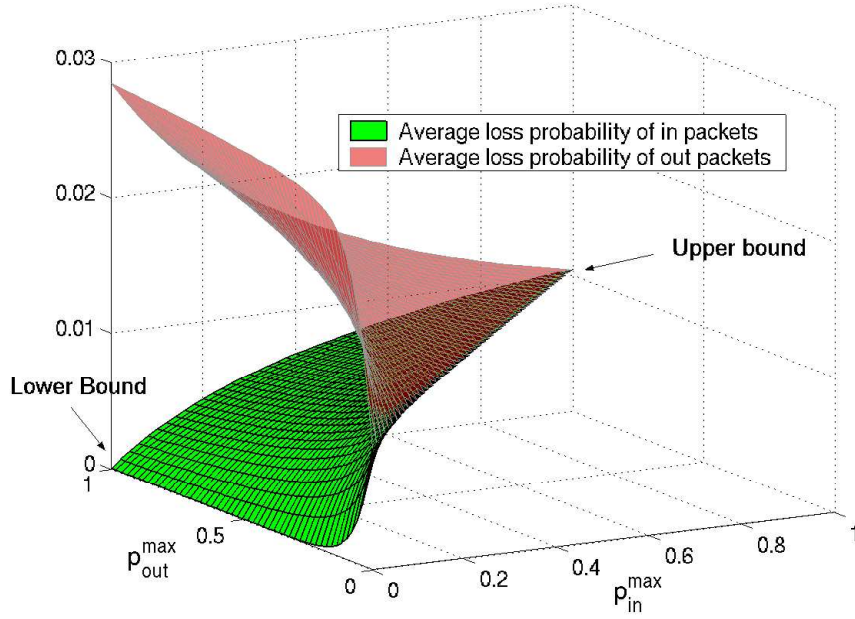


Figure 4.12: Effect of drop probability thresholds on the average loss probabilities

shown that Tail Drop mechanism is particularly suitable for *in* packets to satisfy various QoS constraints. Our method can be extended to handle the case with arbitrarily connected routers.

Appendix A

The expected throughput of *in* packets and *out* packets for a single TCP connection are the following (we omit the superscript i for clarity):

$$T_{in} = \frac{(1 - p_{TO})S_{in} + p_{TO}S_{in}^{\sigma} + p_{TO}R}{(1 - p_{TO})Y + p_{TO}Y^{\sigma} + p_{TO}Z} \quad (4.8)$$

$$T_{out} = \frac{(1 - p_{TO})S_{out} + p_{TO}S_{out}^{\sigma}}{(1 - p_{TO})Y + p_{TO}Y^{\sigma} + p_{TO}Z} \quad (4.9)$$

where

$$S_{in} = \begin{cases} \rho \frac{W}{2} & \text{if } \rho \leq \min(\rho_1, \rho_3) \\ -\frac{W^2}{8} + \rho W + \sigma - \frac{\rho^2}{2} & \text{if } \rho_1 < \rho < \rho_2 \\ 3\frac{W^2}{8} & \text{if } \rho \geq \max(\rho_2, \rho_3) \end{cases} \quad (4.10)$$

$$S_{in}^{\sigma} = \begin{cases} \frac{\rho W^{\sigma}}{2} + \sigma & \text{if } \rho \leq \min(\rho_1^{\sigma}, \rho_3^{\sigma}) \\ S_{in} & \text{elsewhere} \end{cases} \quad (4.11)$$

$$S_{out} = \begin{cases} 3\frac{W^2}{8} - \frac{W\rho}{2} & \text{if } \rho \leq \min(\rho_1, \rho_3) \\ \frac{W^2}{2} - \rho W - \sigma + \frac{\rho^2}{2} & \text{if } \rho_1 < \rho < \rho_2 \\ 0 & \text{if } \rho \geq \max(\rho_2, \rho_3) \end{cases} \quad (4.12)$$

$$S_{out}^\sigma = \begin{cases} 3\frac{W^2}{8} - \frac{\rho W^\sigma}{2} - \sigma & \text{if } \rho \leq \min(\rho_1^\sigma, \rho_3^\sigma) \\ S_{out} & \text{elsewhere} \end{cases} \quad (4.13)$$

$$Y = \frac{W}{2}RTT, \quad Y^\sigma = \frac{W^\sigma}{2}RTT$$

$$R = \frac{1}{1-p_{in}}, \quad Z = RTO \frac{1+p_{in}+2p_{in}^2+4p_{in}^3+8p_{in}^4+16p_{in}^5+32p_{in}^6}{1-p_{in}}$$

$$p_{TO} = \frac{3}{W}$$

$$W = \begin{cases} \frac{\rho\delta + \sqrt{\rho^2\delta^2 + 6p_{out}}}{\frac{3}{2}p_{out}} & \text{if } \rho \leq \min(\rho_1, \rho_3) \\ \frac{4\rho\delta + 2\sqrt{4\rho^2\delta^2 + 2(4p_{out}-p_{in})\left[(\sigma - \frac{\rho^2}{2})\delta + 1\right]}}{4p_{out}-p_{in}} & \text{if } \rho_1 < \rho < \rho_2 \\ \sqrt{\frac{8}{3p_{in}}} & \text{if } \rho \geq \max(\rho_2, \rho_3) \end{cases} \quad (4.14)$$

$$W^\sigma = \begin{cases} \frac{\frac{\rho}{2}\delta + \sqrt{\frac{\rho^2}{4}\delta^2 + \frac{3}{2}p_{out}[\sigma\delta + 1]}}{\frac{3}{4}p_{out}} & \text{if } \rho \leq \min(\rho_1^\sigma, \rho_3^\sigma) \\ W & \text{elsewhere} \end{cases} \quad (4.15)$$

$$\rho_1 = \frac{\sqrt{2\sigma}(2p_{out}+p_{in}) + \sqrt{2\sigma\delta^2 + 2(2p_{in}+p_{out})}}{2p_{in}+p_{out}}, \quad \rho_2 = \sqrt{\frac{8}{3p_{in}}} - \sqrt{2\sigma}, \quad \rho_3 = \sqrt{\frac{3}{2p_{in}}}$$

$$\rho_1^\sigma = \frac{\sqrt{2\sigma\delta+2}}{\sqrt{p_{out}+2p_{in}}}, \quad \rho_3^\sigma = \sqrt{\frac{3}{2p_{in}}}(1-p_{in}\sigma)$$

$$\delta = p_{out} - p_{in}$$

Part II

Overlay Multicast

Chapter 5

Overlay Multicast: A New Multicasting Approach

IP Multicast [36, 39] is a delivery service that can offer tremendous savings in bandwidth for applications that require delivery of the same data to multiple receiver destinations. In IP multicast a source node sends only one copy of the data through the outgoing link without maintaining a list of receivers. IP-multicast-aware routers are responsible for *duplicating* and forwarding the data to any node that wishes to participate in the multicast communication. Hence, physical links that carry the data, and the routers, together form a spanning tree. The spanning tree is constructed by the IP multicast routing protocol based on the network topology, the network state and some optimization objectives.

Several protocols for IP multicast routing and group management have been proposed to support multicast applications. Examples of routing protocols include [39] Distance Vector Multicast Routing Protocol (DVMRP), Multicast Open Shortest Path First (MOSPF), Protocol-Independent Multicast - Sparse Mode (PIM-SM) and Core Based Trees (CBT). These protocols were implemented in the experimental MBONE network [78] and they have been used by several multicast applications [121, 84, 93, 59, 68, 107, 94, 58, 14].

However, numerous architectural and economic complications have prevented a ubiquitous deployment within the Internet [38]. The major issue is that the duplication mechanism in multicast-capable routers make access control and group management a difficult task to handle by network owners. First, denial-of-service attacks by flooding are much easier than in point-to-point connections. Second, routers must maintain per-group routing tables which not only increases the complexity of the routing protocol but also causes scalability issues. Third, providing different levels of service using for example the DiffServ model needs the design of additional protocols to avoid violation of quality of service. In particular, group membership is dynamic, members join and leave the multicast group at any time. New members can receive a quality that is more than required and thus affect existing traffics by consuming non reserved resources [17]. Fourth, inter-domain routing protocols such as Multicast Source Discover Protocol (MSDP) and MultiProtocol Border Gateway Protocol (MBGP) are still not well-defined [3, 38], which slows down ISPs from enabling IP multicast in their domains.

These issues have motivated in part the efforts that explore the implementation of multicast within the application layer upon network overlays: virtual networks that directly connect application-layer-supporting network components to one another by tunneling transmissions across the underlying, unicast-only network. We refer to this scheme as *overlay multicast*¹.

Overlay multicast can provide a low-cost solution to scalably broadcast information to groups of users. However, bandwidth limitations constrain multicast tree fanouts leading to a limitation in the number of receivers and/or high end-to-end delivery delays. Indeed, it is for this reason, Deering [36] and most of the research community advised that multicast functionalities should be implemented at the IP layer. Nevertheless, overlay multicast has many attractive features that make it deployable and commercializable. Overlay Multicast does not need any changes in the current Internet infrastructure and there is no group address assignment problem. Furthermore, there is no need for additional mechanisms for end-to-end reliability or congestion control since they are already supported by unicast protocols. Overlay multicast protocols will focus instead on the design of efficient mechanisms for group management and overlay optimization.

In the next section, we present how overlay multicast works. Then we describe the different approaches of overlay multicast and we outline some design issues. Finally, we study routing algorithms for overlay construction.

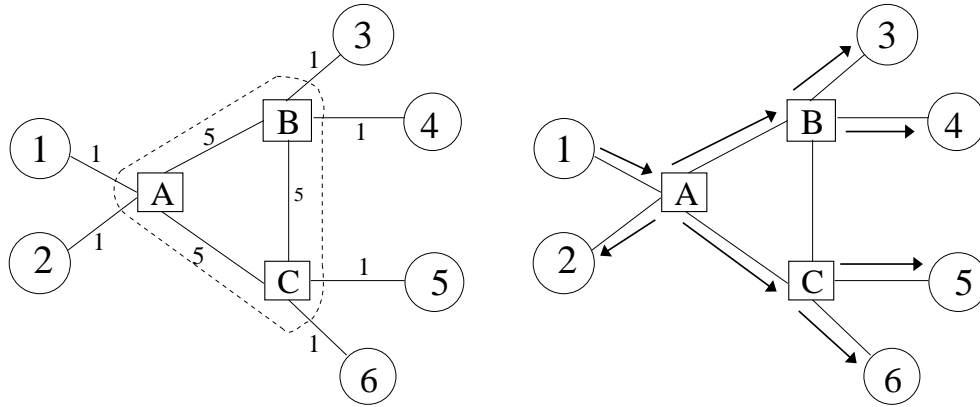
5.1 Overlay Multicast

To illustrate the basic idea of overlay multicast, we use the simple physical network topology of Figure 5.1(a). A, B and C represent edge routers of a network domain. Nodes 1 to 6 are end-systems that can be connected to the routers via modems or DSL. Let us assume that the delay (cost) between any router and its end-system is 1 unit. The delay between any two routers in the network is 5, as indicated in the figure. Now, suppose that node 1 wishes to send real time video at a constant rate r to all the other end-systems. In IP multicast the tree formed is shown in Figure 5.1(b). This tree can be constructed using MOSPF. Figure 5.1(c) shows how the data is delivered using an overlay spanning tree. Each link of the tree corresponds to a unicast path between two end-systems. Each node participating in the forwarding of the data must have enough outgoing bandwidth to maintain all the unicast connections. For example, node 2 needs $2 * r$ units of bandwidth to be able to connect both nodes 5 and 6. Thus the bandwidth of the node constrains its fanout (out-degree).

With overlay multicast, the maximum delay from any node to the source (the depth in the tree) is increased from 7 to 9 compared to the IP multicast scenario. This delay could be worse if end-systems have less amount of bandwidth. For instance, if node 2 has r units of bandwidth, then it can connect to only one other node and the maximum delay will increase as shown in Figure 5.1(d). The overlay tree that can equalize the same delay realized by IP multicast is the one achieved by naive unicast connections, all originated from the source.

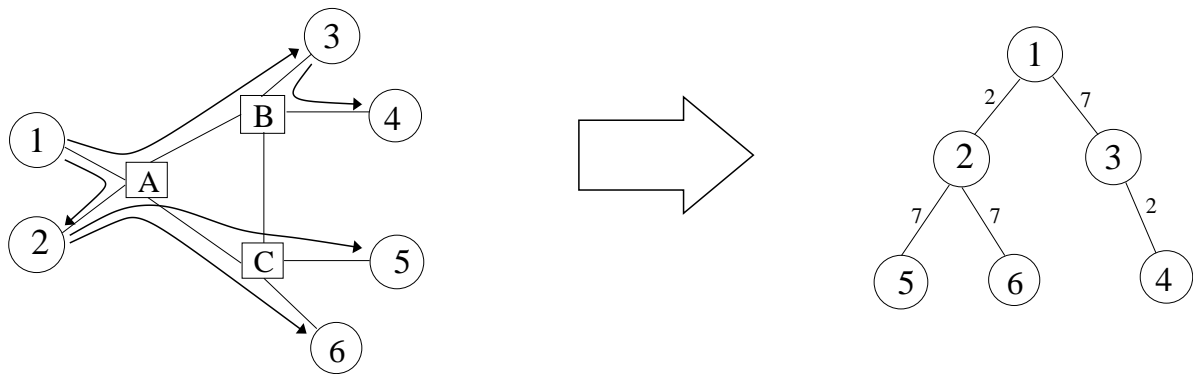
For this network topology, it is easy to find that the trees constructed in Figure 5.1(c) and 5.1(d) minimize the maximum delay given the bandwidth constraints. In a more complicated

¹called also Application Level Multicast

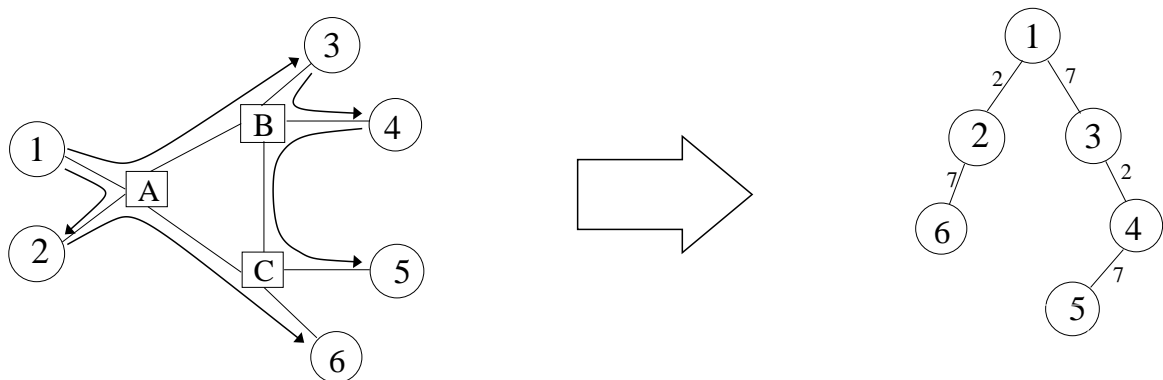


(a) Network scenario

(b) IP multicast scheme



(c) Overlay tree with minimum delay to source, the maximum fanout is 2



(d) Overlay tree with tighter bandwidth constraint for node 2

Figure 5.1: Overlay multicast scheme

network with hundreds of end-systems, it is not obvious to find optimal ("good") trees.

5.2 Overlay Multicast Approaches

Previous work considers two variants of this application layer multicast. The first variant, known as *End-System Multicast (ESM)* forms the overlay out of the very network end-systems that wish to receive the broadcast. The second variant called *Proxy-Based Multicast (PBM)* uses *proxies* at some strategic locations as multicast forwarding points. The best illustration of this architecture is MBONE [78]. An alternative approach that we will study in the next chapter is a hybrid solution where both proxies and end-systems perform forwarding. We call this approach *Proxy-Assisted End-System Multicast (PEM)* [86].

5.2.1 End-System Multicast

In this approach, end-systems connect together over unicast channels to form a multicast tree, rooted at the transmission source in which the end-systems are the nodes and the unicast channels are the edges. ESM is a cheap alternative because there is no need to pay the network or service provider for additional multicast support - the users controlling the end-systems provide all additional functionality necessary. Narada [31, 30], Yoid [53], ALMI [103] and NICE [9] are based on the ESM approach.

However, applicability of ESM to delay-sensitive applications is constrained for two reasons. First, the transmission bandwidth available to end-systems is often limited by the last-mile connectivity technology such as modem, DSL, and cable, limiting the number of copies of a transmission that an end-system can simultaneously forward. Hence, interior nodes of end-system multicast trees tend to have lower fanout resulting in trees of greater depth and delivery delay. Second, transfer delays across this "last mile" are often a significant fraction of a connection's overall end-to-end delay, ranging between 20ms and 150ms depending upon the last mile technology. These large delays further magnify the delay penalties formed from the longer chains of end-systems.

5.2.2 Proxy-Based Multicast

A proxy is a high-bandwidth multicast forwarding device that is provided by the owner of the network or an organization. It can be a router or an end-system that can be strategically placed within the network. End-systems connect directly to the nearest proxy via traditional unicast to receive the broadcast. Proxies can be attached directly to high-speed lines. In comparison to end-system multicast, because proxy transmission quantities are not constrained by last-mile bandwidth limitations, trees formed from proxies are flatter and wider with lower-delay edges. Overcast [73], Akamai [1], Inktomi [64] and RMX [27, 28] uses this approach.

5.3 Overlay Multicast Design

In order to provide efficient methods to forward data in a self-organized overlay, protocols must be designed to build and maintain the overlay multicast tree. Essentially, the design of an overlay multicast protocol includes two components: a group management component and an overlay optimization component.

5.3.1 Group Management

The group management component is responsible for constructing the tree that connects all end-systems participating in the multicast session. Two basic methods were proposed for construction of overlay spanning trees. The first method, called tree-first [53], consist of constructing the tree directly, i.e. nodes are attached to the tree incrementally by selecting a parent for each node. An appropriate algorithm is also deployed to avoid producing loops in the target tree.

The second method, called mesh-first [30], is to construct the tree in two steps. First, a strongly connected graph termed a mesh is constructed to connect members. The links of the mesh corresponds to unicast paths between members, in the extreme case the mesh may include $N * (N - 1)$ overlay links if the number of members is N . Second, a tree rooted to the source is constructed by simply adapting standard routing algorithms.

This component also handles dynamic group membership which involves connecting new arrival members and also connecting members who were connected via an other member who left the multicast session or who broke down. Notice that with the PBM approach the leave operation do not involve connecting sub-trees because a proxy is allowed to leave the tree only if all its children leave before.

5.3.2 Overlay Optimization

The overlay optimization component ensures that the overlay tree is of "good" quality and remains good over time. It maintains all the performance metrics and the optimization objectives necessary for the routing protocol to construct the tree. These optimization objectives depend entirely on the application requirements. For instance, file transfer applications are more interested in the bandwidth than the transfer delays. In contrast, conferencing applications need to keep small transmission delays. Since network delays and available bandwidth are dynamic, the optimization component must also update the performance metrics, using active measurements, to adapt the tree to dynamic changes.

5.4 Routing in Overlay Multicast

The overlay tree is constructed by the multicast routing protocol taking into account the network resources and some optimization objectives. The multicast routing or the tree construction is usually formulated as an optimization problem subject to some constraints. The constraints reflect the network resources available and the needs of the multicast application itself such as

end-to-end delay from the source to any receiver in the tree, minimum reception bandwidth, the loss probability, the jitter and/or some combination of them.

In overlay multicast, the network can be modeled as a complete graph where links correspond to unicast paths. As overlay networks communicate across tunnels implemented at the transport layer, it is possible for an overlay node to communicate directly with any other overlay node in the network. The graph is directed or undirected depending on whether the target tree is source-specific or shared. In the case of source-specific trees, a tree is built for each node that has data to send (source node). The tree connects only the source and its receivers. There, we use the *depth* to reflect the application needs in terms of delay. In the shared tree case, a single tree connects all the nodes, and consequently, we use the *diameter* of the tree. In all cases, the bandwidth and the transmission rate of the multicast applications determine the fanout (degree) of each node on the graph.

Previous work on building optimal multicast trees have often been in the context of network layer multicast where node fanouts are restricted only by their degree within the underlying graph topology and where inclusion of router nodes in the tree is optional. There, the general optimization problems are variants of Steiner-Tree problems and are also NP-complete. Several papers have developed heuristic approximations [77, 102, 133, 122, 106] or ratio-bounded approximations [26] to these optimization problems. However, these heuristics are not applicable directly to overlay multicast since there is no straightforward way to account for degree constraints of nodes in a tree that are bounded by its degree in the underlying connectivity graph.

Multicasting problems and the algorithms that can be used to solve them have been classified in the context of network layer multicast [123]. In overlay multicast, we can distinguish two classes of problems, bandwidth optimization and delay optimization.

5.4.1 Bandwidth Optimization

Constructing an overlay optimized for both delay and bandwidth can be complicated [124]. Some overlay multicast protocols choose the bandwidth as the primary performance metric. This could be suitable for non interactive applications where small delays are of less importance. Overcast [73] and Narada [30] use this method to build the overlay tree.

Overcast uses the tree-first approach. Each node is attached to the tree to maximize the available bandwidth to the root. The algorithm starts by considering the root as a possible candidate to be the attachment point. Then, the node is attached to the root of the tree if the available bandwidth of the root is larger than all its children. Otherwise, the child that has the largest available bandwidth becomes the new candidate. If many candidates are suitable, then the one closest in terms of network hops to the node is selected. This procedure is repeated until it finds the best attachment node. As a consequence, the node can be placed far away from the root (Figure 5.2(a)).

Narada uses a slightly different routing technique since it uses the mesh-first approach. The bandwidth optimization is incorporated in the distance vector protocol that runs over the mesh using a variant of the widest path first algorithm introduced in [124]. Each overlay link in the mesh is assigned an estimation of the available bandwidth. Then, every node tries to pick the

widest path to every other node. In the case of multiple paths with the same bandwidth, the path with the lowest delay is chosen (Figure 5.2(b)). This algorithm is similar to the Bellman-Ford algorithm that computes the shortest-path using distance vectors. Thus, the time complexity involved is $O(nm)$ where n is the number of nodes and m is the number of links.

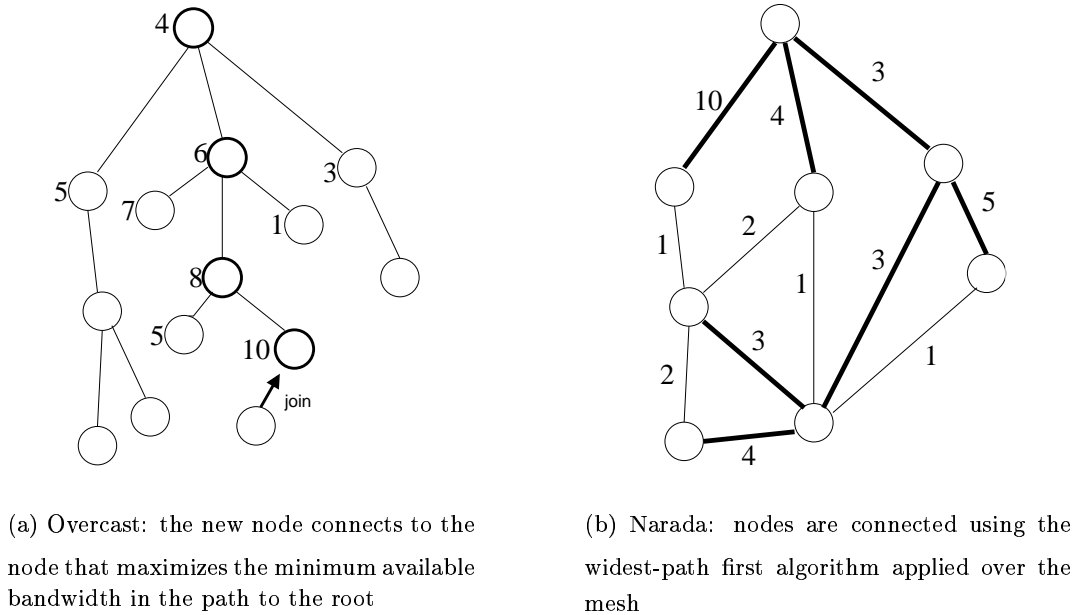


Figure 5.2: Bandwidth optimization

5.4.2 Bandwidth Constrained Delay Optimization

Bounded-degree multicast has also been explored in the context of network layer multicast. Some heuristics of building shared (non-specific-source) minimum spanning trees that minimize aggregate edge costs (instead of minimizing the delay from a specific source) were developed in [12, 11]. The heuristics are variants of the shortest path heuristic introduced in [118].

The bandwidth (degree) constrained delay optimization problem in the context of overlay multicast has been studied in [113]. They formulated a problem using an undirected complete graph with a degree bound for each node. They call the problem the Minimum Diameter, Degree Bounded Spanning Tree (MDDBST). The goal is to minimize the diameter of the tree while satisfying the bandwidth limitations of nodes (proxy servers). They propose a heuristic which is inspired from Prim's algorithm [35]. The algorithm, which is greedy, starts by selecting a node as the root of the tree T . Each node v maintains a measure $dia(v)$ of the longest path to any other node in the the current tree T . Then, at each step when adding a new node to the existing tree, the node that has the smallest $dia(v)$ is selected. When updating $dia(v)$, only nodes that still have available bandwidth participate in the computation. The total running time of this greedy algorithm is $O(n^3)$, where n is the number of nodes in the graph (number of

participants).

This algorithm can not ensure that the diameters of the constructed trees are small enough, since it considers the bandwidth as a constraint. If we place a node with a small $dia(v)$ but with small available bandwidth then the paths involving this node may become very long in next steps resulting in a long diameter tree. Also, the nodes may have unbalanced available bandwidth at the end of the algorithm. For example a situation where leaf nodes have large available bandwidth might be undesirable.

5.4.3 Delay Constrained Bandwidth Optimization

In section 5.4.1, the bandwidth is optimized by maximizing the minimum (available) bandwidth from a given node to the source through the overlay tree. Another approach is to maximize the minimum available bandwidth for each node in the tree [113, 114]. The goal is to distribute the available bandwidth in the most balanced way in order to construct as many trees as possible using the same set of nodes which allows for multiple sessions to share the same network. We will study this issue in more detail in chapter 8.

Since bandwidth optimization can lead to deep trees, it is interesting to set some "acceptable" bound on the delay. Computation of delay-bounded paths that minimize a monotone or additive metric, is described in [56] without covering the case of fanout constraints.

In [113], a bound is set for the diameter of the tree and the problem is introduced as the Bounded diameter, residual-balanced spanning tree (BDRBST). Two heuristics were developed to resolve this optimization problem. The first [113], extends the heuristic MDDBST of the previous section by adding a constant M to play a role of a balance factor. At each step of the algorithm, when adding a new node, instead of selecting the one with the shortest $dia(v)$, the first M nodes with the shortest $dia(v)$ are selected, then the node that maximizes the minimum available bandwidth of these M nodes and their parent nodes is selected. If $M = 1$ then the algorithm reduces to the MDDBST algorithm. If $M = +\infty$ (the total number of nodes) then the algorithm solves only the bandwidth optimization problem. The greedy nature of this algorithm prevents it from achieving the best balanced tree. Besides, it is not obvious to find the best value of M to control the trade-off between the bandwidth and the delay.

The second method introduced in [114] uses a more general approach to construct a balanced tree. The algorithm starts by determining the "ideal" degree of each node in the target tree with respect to the same objective of maximizing the minimum available bandwidth. This set of degrees defines a *balanced degree allocation* (BDA). The BDA represents a sort of new degree constraint for the rest of the algorithm. Next, the algorithm tries to find a tree in which each node has the degree specified by the BDA while satisfying the constraint on the diameter. At this step, any algorithm can be used to add incrementally the links (pair of nodes) to construct the tree using a selection rule. For example, one can use the algorithm that selects simply the closest pair of nodes. Also, we can apply the MDDBST algorithm to achieve this step. However, it is possible that a tree that satisfies the constraint on the diameter will not be found. In that case the degree allocation must be relaxed in some way and the same procedure must be repeated

iteratively. After each iteration the BDA must be relaxed again if no tree is found. As a result, if the delay constraint is tight, then the computation of the “best” balanced degree allocation does not guarantee that the final tree will have effectively balanced available bandwidths. One method for relaxation [114] is to change the BDA using geographical information by increasing the degree of central nodes first.

5.5 Conclusion

With the exception of [113], studies to date in the area of application layer multicast have been experimental in nature for both end-system multicast and Proxy-Based Multicast.

In the next two chapters, our goal is to design algorithms of overlay construction, guided by theoretical study. We will provide a formulation of optimization problems, and aim at an understanding of the complexities involved. We will use simulations to evaluate the performance of the algorithms in various scenarios. We note that we do not provide protocols that are ready for deployment. Indeed, we do not focus on several practical issues such as building distributed versions of the algorithms. More specifically, we first focus on a specific problem of the minimization of proxy costs in a hybrid proxy/end-system environment. We then study the problem of sharing network resources among multiple overlay multicast sessions.

Chapter 6

Bounding Delay in Proxy-Assisted End-System Multicast

In this chapter, we focus on the needs of distributed applications such as teleconferencing, distributed gaming, chat rooms, and small-scale live concerts or sporting events where the number of receivers is in the tens or hundreds. For these kinds of applications, *low-latency* delivery is of paramount importance, as is keeping session costs to a minimum. We consider applications designed to cope with delays up to some Δ . Transmission below this delay can be achieved upon Proxy-Based Multicast with high bandwidth proxies. However, we assume that each proxy charges per copy of the transmission that it forwards. Hence, our goal is to restrict the number of transmissions that emanate from proxies by using end-systems to perform the multicast wherever possible and still meet the end-to-end delay bound requirements of the application for all session participants.

To this end, we first provide a formal description of a hybrid proxy/end-system network with delay-constrained multicast applications. Second, we provide an algorithm to build the multicast tree and formally prove its optimality in a fully-connected overlay network with uniform-length edges. Then, we adapt this algorithm into a heuristic for the case of variable-delay edges. Finally, we evaluate the performance of the algorithm for simulated transit-stub networks with variable-delay edges.

We develop a simple graphical model to explore algorithms and heuristics in a theoretical context in which nodes represent multicast-capable agents (proxies or end-systems) and edges represent routes between these agents. Any “acceptable” multicast tree built on top of this graph satisfies three requirements: a) it must connect all end-system nodes (but need not connect all proxies) to the source, b) the delay along the path from the source to any end-system must be below the specified delay bound, and c) the number of children of a node must fall beneath the bound imposed by the node bandwidth constraint.

We provide an optimal algorithm for the case in which the delays between all pairs of nodes in the graph are identical, but where fanouts from the various nodes can differ due to the variety in access bandwidths available to the nodes. This uniform-delay assumption could be appropriate for networks where the most significant delays are due to high transfer delays of identical last-mile technologies, as an approximation for delay in systems where precise node-to-node delays

are unavailable, or where all node guarantees only that they will forward a packet to the next hop on the overlay within a fixed time bound. Next, we consider environments in which delays between nodes need not be uniform. Here, the optimization problem is NP-hard.

We address the challenge of finding an efficient, low-complexity solution by extending our optimal algorithm to a heuristic that quickly identifies a “good”, but not necessarily optimal tree. The heuristic contains a tunable parameter that allows us to adjust the relative importance given to bandwidth constraints of nodes versus delay constraints of edges. We evaluate the performance of the heuristic through simulation on randomly generated transit-stub topologies and consider cases where proxy placement is restricted to within the backbone, restricted within the stub networks, restricted to access points, or is unrestricted. We evaluate our heuristic in two ways. First, we compare the maximum end-system delay of proxy-free trees built by our heuristic to those built by the heuristic developed in [30]. Simulation results show that our heuristic provides a 25% reduction in delay. Next, we demonstrate the expected cost (where cost equals the number of edges extending from proxies) of trees built by the heuristic to achieve tighter delay bounds.

6.1 Architecture and Model

In this section, we present a more formal definition of our network model and formally pose the optimization problem. However, we first begin by giving a high level description of the problem setting. We consider a set of end-system nodes in which one node in particular wishes to multicast information to the other participating nodes. This is accomplished by constructing a *multicast overlay tree* where end-system nodes forward the transmission to (several) other participating end-system nodes via unicast connections. These nodes (including the source) have limited bandwidth capabilities, such that they must form trees in which each node is only required to forward data to a small set of other nodes.

The sessions that we consider here require that the data emanating from the sender must reach all session participants within some fixed amount of time. For cases where transmission delays between end-systems are difficult to predict accurately, or where the delay results from transmission over last-mile technologies, it is desirable to bound the number of hops in the tree that must be traversed to deliver data to an end-systems.

To assist these sessions toward meeting delay bounds, the network provides *forwarding proxies*. These are nodes with access to high bandwidth levels. The multicast session can draw upon these proxies to forward data within the delay constraints to a much larger set of end-systems. However, the network charges a price for each proxies’ services that is an increasing function of the number of copies that the proxy is asked to forward.

For example, Figure 6.1(a) depicts a set of end-system nodes with a particular node labeled S as the data source. S wishes transmit data with low delay to a set of end-system nodes, but the bandwidth limitations imposed on each node (including S) restricts it to simultaneously forwarding at most two copies of the transmission. Hence, a minimal-depth multicast overlay tree in this example has depth 3: the tree depicted in Figure 6.1(b) is an example of one such tree. Figure 6.1(c) depicts the same set of end-system nodes in a network where two high bandwidth proxies are also made available. The one on the left can simultaneously forward data to 6 end-

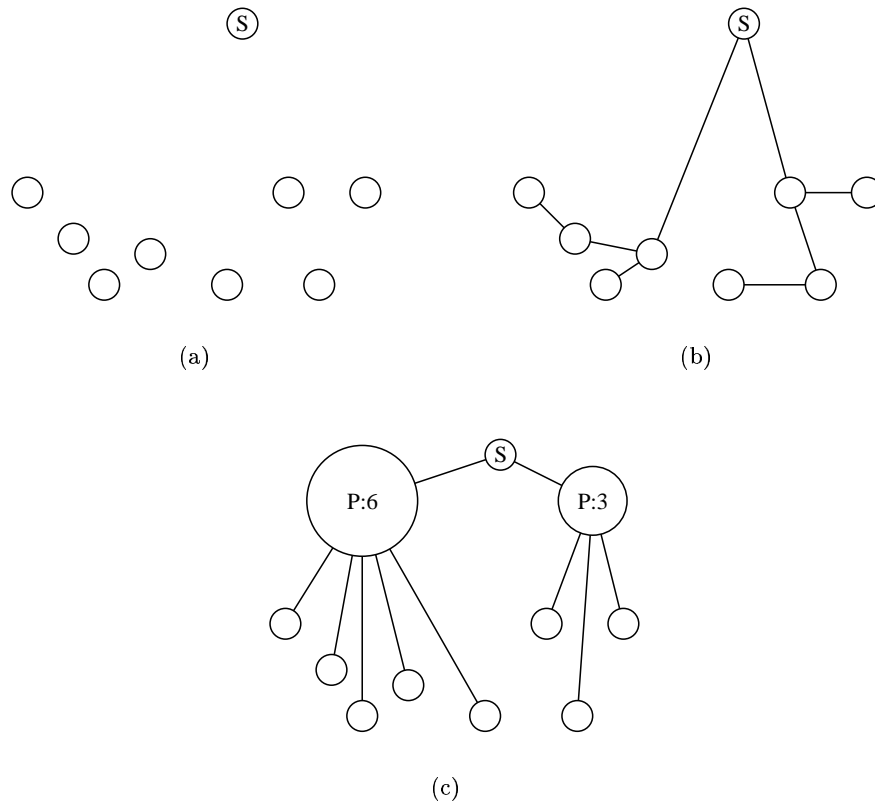


Figure 6.1: Reducing Delay with Forwarding Proxies

systems, and the one on the right can simultaneously forward data to 3 end-systems. By utilizing the proxies to respectively forward data directly to 5 and 3 end-system participants, it is possible to build a multicast overlay tree with depth 2. There is no tree that can be built with depth 2 in which the number of transmissions that emanate from the proxies is smaller. Hence, the tree depicted here is a minimum cost tree when a restriction is applied that trees must have depth no more than two.

We now state our model of the network in a more formal manner. Let $G = (s, N, P, E)$ be a network consisting of a source node s , a set of end-system nodes N , a (possibly empty) set of proxy nodes P , and a set of edges E such that an edge $e = \langle n_1, n_2 \rangle \in E$ exists between each pair of nodes $n_1, n_2 \in \{s\} \cup N \cup P$ (i.e., the overlay network is fully connected). Let $d(\langle n_1, n_2 \rangle)$ represent the end-to-end delay from n_1 to n_2 . In addition, our assumption that bandwidth rates are constrained by the last-mile hop translates to the delays along the edges and the bandwidth availabilities between pairs of nodes being independent of the respective delays and bandwidth availabilities at other nodes. The fact that the actual paths represented by these network edges share links in common is of no consequence since these links do not impose bandwidth constraints in our model.

Definition 6.1 (Fanout Constraints) A *fanout constraint function (FCF)*, $f()$, is a

function that maps each node $n \in \{s\} \cup N \cup P$ to a non-negative integer. $f(n) = i$ implies that node n has sufficient bandwidth capabilities to forward session data to at most i other nodes.

We assume that given the transmission rate of the supported session, a node n can determine $f(n)$. For instance, if a node n 's bandwidth capability is r and a session is to be transmitted at rate ρ , then $f(n) = \max\{\lfloor (r - \rho)/\rho \rfloor, 0\}$.¹ We assume proxies have access to larger amounts of bandwidth than end-systems, such that $f(p) > f(n)$ for most proxies p and end-systems n . We will also use FCFs to artificially limit potential fanout from proxies, i.e., we will construct FCFs $f_1()$ where $f_1(n) = f(n)$ for $n \in \{s\} \cup N$ and $f_1(p) \leq f(p)$ for $p \in P$.

Upon G , we wish to build a tree $T = \{s, N, P_T, E_T\}$ formed from nodes $\{s\} \cup N \cup P_T$ where $\emptyset \subseteq P_T \subseteq P$ and $E_T \subseteq E$ are the edges. We define several functions that describe several relevant tree properties:

- $\Pi_n(\mathbf{T})$: the parent of node n in tree T .
- $c_n(\mathbf{T})$: the number of child nodes of n in tree T , i.e., $c_n(T) = |\{n' : \Pi_{n'}(T) = n\}|$.
- $D_n(\mathbf{T})$: the delay from s to node n in tree T where $D_s(T) = 0$ and $D_n(T) \equiv D_{\Pi_n(T)}(T) + d(\langle \Pi_n(T), n \rangle)$.
- $D_{\max}(\mathbf{T}, \mathbf{S}) = \max_{n \in S} D_n(T)$, i.e., the maximum depth in T of any nodes in $S \subseteq T$. For conciseness, we define $D_{\max}(T) \equiv D_{\max}(T, T)$ to be the depth of the tree.

Definition 6.2 (Legal Connectivity) We say a tree T is **legally connected with respect to FCF $f()$** if s is the root of T , $N \subseteq T$, and $c_n(T) \leq f(n)$ for all $n \in T$, i.e., all nodes of N belong to T and no node's maximum degree constraint under $f()$ is violated.

Tree Cost: Let $A(T)$ be the cost for the session to utilize tree T . We restrict our attention to cost functions of the form $A(T) = \sum_{n \in P} g(c_n(T))$, where g is a non-decreasing, concave or linear function. Under such a cost function, the increase in cost to add an additional transmission from a proxy does not increase. This covers cost functions of the form $g(i) = 0$ for $i = 0$ and $g(i) = \mathcal{C}_1 + \mathcal{C}_2 i$ otherwise, i.e., the proxy provider charges \mathcal{C}_1 for each proxy provided to the session plus \mathcal{C}_2 for each copy of the session transmitted from that proxy.

Optimization Problem: The formal description of our optimization problem is as follows: Let Δ be an application delay bound. Let $f()$ be the FCF imposed by the application on nodes $\{s\} \cup N \cup P \in G$ and let $A()$ be the proxy cost function. We wish to find a tree, $T_{\min} = (s, N, P_{T_{\min}}, E_{T_{\min}})$ where

- T_{\min} is legally connected w.r.t. $f()$.
- $D_{\max}(T_{\min}, N) \leq \Delta$.

¹Here, we subtract ρ from the numerator to allow sufficient bandwidth for the node to receive session data.

- $A(T_{\min}) \leq A(T)$ for all other trees T that are legally connected w.r.t. $f()$ and where $D_{\max}(T, N) \leq \Delta$.

6.2 Uniform Edge Overlays

In this section, we present an algorithm (called FindMinCost) for use in a network where the end-to-end delay between all pairs of nodes is uniform (we assume that $\forall n_1, n_2 \in \{s\} \cup N \cup P, d(\langle n_1, n_2 \rangle) = 1$).

We prove that, given a delay bound, Δ , the algorithm finds a multicast tree that minimizes the cost to connect the source to all session receivers along paths with a delay less than Δ . This algorithm is developed in three steps. In the first step, we present an algorithm (called MinDepth) that, for a given FCF, $f()$, computes a minimum-depth tree rooted at s . This initial algorithm does not distinguish between proxy nodes and receiver nodes, and hence the tree does not necessarily have minimal cost. Next, we apply the theory of majorization to construct an algorithm (called FindBestProxyTree) that computes a minimum-depth tree that does not exceed a fixed cost, C . FindBestProxyTree is implemented by selecting an appropriate FCF and then applying MinDepth. Last, we construct FindMinCost by choosing various values for C until a cost for which a tree exists with minimum depth no larger than Δ is found.

6.2.1 Minimizing Fanout-constrained Tree Depth

We begin by presenting Algorithm MinDepth which computes a minimum depth tree w.r.t. FCF $f()$ on G . This algorithm solves exactly the bandwidth constrained delay optimization problem (introduced in the previous chapter) in the particular case of uniform edges. The algorithm essentially puts nodes with highest potential fanout (i.e., largest $f(n)$) closer to the source:

Algorithm 6.1 *MinDepth*($G, f()$)

- (1) Let the source node be n_0 and order the nodes in G as $n_1, n_2, \dots, n_{|N|-1}$ such that $f(n_i) \geq f(n_j)$ for all $1 \leq i < j < |N|$.
- (2) $i = 1, j = 0$
- (3) While $i < |N|$
- (4) Set $\Pi_{n_i}(T_{\min}) = n_j$
- (5) $i ++$
- (6) If $c_{n_j}(T_{\min}) = f(n_j)$ then $j ++$
- (7) end loop
- (8) return T_{\min}

Lemma 6.1 *Algorithm MinDepth generates a minimum-depth legally connected tree w.r.t. $f()$.*

Proof:

Let L_T^i be the set of nodes within tree T that reside at depth less than or equal to i . We begin by proving the following claim:

Claim 6.1 $|L_T^i| \leq 1 + \sum_{n \in L_T^{i-1}} f(n) - |L_T^{i-1}| = 1 + \sum_{n \in L_T^{i-1}} (f(n) - 1)$.

Proof: Every node in the subtree formed from the nodes in L_T^{i-1} has one parent with the exception of the root, which has no parents. The sum of fanouts of nodes in L_T^{i-1} is $\sum_{n \in L_T^{i-1}} f(n)$, and with $|L_T^{i-1}| - 1$ edges “used” to connect a node to its parent, there remain at most $\sum_{n \in L_T^{i-1}} f(n) - |L_T^{i-1}| + 1$ edges that can connect nodes of depth i to nodes of depth $i - 1$ in T . \blacksquare

We next show by induction on i that for any legally connected tree T w.r.t. $f()$, $|L_{T_{\min}}^i| \geq |L_T^i|$ for all $i < D_{\max}(T_{\min})$. For $i = 0$, the result clearly holds, since $|L_{T_{\min}}^0| = |L_T^0| = 1$, i.e., just the root node. Assume that $|L_{T_{\min}}^{i-1}| \geq |L_T^{i-1}|$, and let $S(j)$ be the set of j nodes first attached to the tree by MinDepth. Since MinDepth places nodes of largest fanout at the minimum depths of the tree, it follows that $\sum_{n \in S(j)} f(n) \geq \sum_{n \in S'} f(n)$ for any other set of nodes S' where $|S'| = |S(j)| = j$. By the construction of T_{\min} using Algorithm MinDepth, no nodes in $L_{T_{\min}}^{i-1}$ are leaves, and it must be the case that $f(n) - 1 \geq 0$ for all $n \in L_{T_{\min}}^{i-1}$. Choosing $j = |L_T^{i-1}|$, since $|L_{T_{\min}}^{i-1}| \geq |L_T^{i-1}|$, we have $S(j) \subseteq L_{T_{\min}}^{i-1}$ and

$$\begin{aligned}
& \sum_{n \in L_{T_{\min}}^{i-1}} (f(n) - 1) \\
&= \sum_{n \in S(j)} (f(n) - 1) + \sum_{n \in L_{T_{\min}}^{i-1} \setminus S(j)} (f(n) - 1) \\
&\geq \sum_{n \in S(j)} (f(n) - 1) \geq \sum_{n \in L_T^{i-1}} (f(n) - 1). \tag{6.1}
\end{aligned}$$

Furthermore, since MinDepth does not add nodes at depth i until $c_n(T_{\min}) = f(n)$ for all nodes n where $D_n(T_{\min}) < i - 1$, we have that

$$\begin{aligned}
|L_{T_{\min}}^i| &= 1 + \sum_{n \in L_{T_{\min}}^{i-1}} f(n) - |L_{T_{\min}}^{i-1}| \\
&= 1 + \sum_{n \in L_{T_{\min}}^{i-1}} (f(n) - 1) \\
&\geq 1 + \sum_{n \in L_T^{i-1}} (f(n) - 1) \tag{6.2}
\end{aligned}$$

$$\geq |L_T^i|, \tag{6.3}$$

where (6.2) holds due to (6.1) and (6.3) holds due to Claim 6.1, completing the proof by induction (for all $i < D_{\max}(T_{\min})$). The final result follows from the fact that for any $i < D_{\max}(T_{\min})$, $|L_T^i| \leq |L_{T_{\min}}^i| \leq |N|$. Thus, not all nodes of T can lie at a depth less than $D_{\max}(T_{\min})$. \blacksquare

6.2.2 Applying Majorization to FCFs

We next develop an algorithm that inserts proxy nodes within a tree that minimizes tree depth while keeping costs below a given bound, C . Before introducing the algorithm, we introduce a majorization technique and prove an important property of majorization that is critical in demonstrating the correctness of our algorithm.

Definition 6.3 *Let $f_1()$ and $f_2()$ be two different FCFs in G . We say that $f_1()$ **majorizes** $f_2()$ and write $f_1() \succ f_2()$ if there exist two orderings of the nodes in G , $\alpha_1, \dots, \alpha_m$ and β_1, \dots, β_m such that all the following hold:*

- $f_1(\alpha_i) \geq f_1(\alpha_j)$ and $f_2(\beta_i) \geq f_2(\beta_j)$ for $i < j$.
- For all $j > 0$, $\sum_{i=1}^j f_1(\alpha_i) \geq \sum_{i=1}^j f_2(\beta_i)$.
- $\sum_{i=1}^m f_1(\alpha_i) = \sum_{i=1}^m f_2(\beta_i)$.

Lemma 6.2 *Let $f_1()$ and $f_2()$ be two different maximum fanout assignments in which $f_1() \succ f_2()$. If T_0 is a legally connected tree in G with respect to $f_2()$, then there exists a tree T_1 that is legally connected with respect to $f_1()$ where $D_{\max}(T_1) \leq D_{\max}(T_0)$.*

In other words, if the total number of edges extending from all nodes must remain fixed, the minimum depth of a tree is reduced by having some nodes with a very large number of edges and others with a very small number of edges than when all nodes have roughly the same number of edges.

Proof: Let $\alpha_1, \dots, \alpha_m$ be an ordering of the nodes in G in which $f_1(\alpha_{i+1}) \leq f_1(\alpha_i)$ and β_1, \dots, β_m be an alternate ordering in which $f_2(\beta_{i+1}) \leq f_2(\beta_i)$. We construct T_1 as follows: first, create a tree T_2 that is legally connected w.r.t. $f_2()$ using Algorithm 6.1. By Lemma 6.1, $D_{\max}(T_2) \leq D_{\max}(T_0)$. Next, form a tree T_3 in which node α_i connects to node α_j iff β_i connects to β_j in tree T_2 . Since the trees have isomorphic connectivity structure, $D_{\max}(T_3) = D_{\max}(T_2)$. Some nodes in T_3 may violate fanout constraints of $f_1()$ such that T_3 is not legally connected w.r.t. $f_1()$. We convert T_3 into a legally connected tree w.r.t. $f_1()$, T_1 by detaching subtrees rooted at nodes n where $c_n(T_3) > f_1(n)$ and re-attaching them to nodes n' where $D_{n'}(T_3) \leq D_n(T_3)$ and $c_{n'}(T_3) < f_1(n')$. The fact that a sufficient number of available edges exist at lower depths of the tree follows from the condition that $f_1() \succ f_2()$ and hence for all $j > 0$, $\sum_{i=1}^j f_1(\alpha_i) \geq \sum_{i=1}^j f_2(\beta_i)$ and that in trees constructed by Algorithm MinDepth, the depth of nodes of larger fanout is no more than those of lesser depth. The transition from T_3 to T_1 involves only the shifting of some subtrees closer to the source. Hence, $D_{\max}(T_1) \leq D_{\max}(T_3) = D_{\max}(T_2) \leq D_{\max}(T_0)$. ■

6.2.3 Minimizing Proxy Involvement

We now design an algorithm that calls Algorithm MinDepth to generate the minimum-depth tree with cost no more than a given C . We do this by restricting the number of children that proxies are permitted beyond the restrictions imposed by their fanout constraints by choosing a particular FCF for proxies when computing the minimum-depth tree. Lemma 6.2 uniquely determines the FCF to use within the algorithm:

Algorithm 6.2 *FindBestProxyTree*($G, C, f()$)

- (1) Order proxy nodes a_1, \dots, a_m such that $f(a_i) \geq f(a_{i+1})$.
- (2) Construct $f_1()$ such that $f_1(n) = f(n)$ for all $n \in \{s\} \cup N$ and recursively compute $f_1(a_i) = \max_{0 \leq k \leq f(a_i)} \{k : \sum_{j=1}^{i-1} g(f_1(a_j)) + g(k) \leq C\}$.
- (3) return *MinDepth*($G, f_1()$)

Lemma 6.3 For any pair of FCFs, $f_1(), f_2()$, if for all $n \in G$, $f_1(n) \leq f_2(n)$, then the depth of the minimum-depth tree that is legally connected w.r.t. $f_2()$ is less than the depth of the minimum-depth tree that is legally connected w.r.t. $f_1()$.

Proof: Any tree that is legally connected w.r.t. $f_1()$ is also legally connected w.r.t. $f_2()$. ■

Lemma 6.4 Let T_0 be the tree generated by Algorithm 6.2. Then $A(T_0) \leq C$. Furthermore, if a tree T exists in which $A(T) = C$ and $D_{\max}(T) \leq d$, then $D_{\max}(T_0) \leq d$.²

Proof: To prove this Lemma, we will make use of the following additional property of majorization: If $f_1() \succ f_2()$ and $g()$ is a linear or concave non-decreasing function, then $\sum_n g(f_1(n)) \leq \sum_n g(f_2(n))$.

Next, consider any general non-decreasing, linear or concave function, $g()$. Again, let T be any tree where $A(T) = C$. Let $f_2(n) = f(n)$ for $n \in \{s\} \cup N$ and $f_2(n) = c_n(T)$ for $n \in P$. Let k be the aggregate number of children of proxies in T (i.e., $k = \sum_{p \in P \cap T} c_p(T) = \sum_{p \in P} f_2(p)$). Let C' be the cost necessary such that Algorithm 6.2 builds $f_1()$ such that $\sum_{i=1}^m f_1(a_i) = k$, and let T_1 be the tree built by Algorithm 6.2 with this cost. By construction, $f_1() \succ f_2()$, such that (by Lemma 6.2) $D_{\max}(T_1) \leq D_{\max}(T)$. Also, the majorization property gives that $C' = \sum_{i=1}^m g(f_1(a_i)) \leq \sum_{i=1}^m g(f_2(\beta_i)) = C$. Since increasing C' might increase but never decreases $f_1(n)$, by Lemma 6.3, increasing the maximum cost from C' to C can only result in a decrease in depth of the tree produced by Algorithm 6.2, we have that $D_{\max}(T_0) \leq D_{\max}(T)$. ■

²Note that it is possible that $A(T_0) < C$ if not all outgoing proxy edges permitted under $f_1()$ are used, and hence, we cannot claim that a tree of cost C exists.

In particular, the above algorithm covers the case where $g(i) = i$, i.e., session cost for a tree T equals sum of the number of children of proxies. The final algorithm developed in this section calls `FindBestProxyTree` with different values of C until the smallest C is identified within which a tree of depth less than Δ can be constructed.

Algorithm 6.3 *FindMinCost*($G, f(), \Delta$)

- (1) Set $C_{\min} = 0, C_{\max} = \sum_a f(a)$
- (2) While $C_{\max} - C_{\min} > 1$
- (3) $C = C_{\min} + \lfloor (C_{\max} - C_{\min})/2 \rfloor$
- (4) $T = \text{FindBestProxyTree}(G, C, f())$ // which in turn calls *MinDepth*()
- (5) if $D_{\max}(T) > \Delta$
- (6) $C_{\min} = C$
- (7) else $C_{\max} = C$
- (8) end if
- (9) end loop
- (10) return *FindBestProxyTree*($G, C_{\max}, f()$)

Theorem 6.1 *Algorithm FindMinCost() finds the minimum cost legally connected tree T w.r.t. FCF $f()$ where $D_{\max}(T) \leq \Delta$ and does so in time $O(n \log^2 n)$, where $n = |s| \cup N \cup P$.*

Proof: Correctness of the algorithm follows trivially from Lemmas 6.4 and 6.3, plus the observation that C_{\min} is never set to a cost for which a legally connected tree w.r.t. $f_1()$ exists that has depth of Δ or less, and that C_{\max} is only set to costs for which such a tree does exist. Algorithm *MinDepth* runs in time $O(n \log n)$ to sort the nodes as a function of FCF $f()$. Building the tree takes $O(n)$ time. *FindBestProxyTree* takes $O(n)$ time to generate $f_1()$ prior to calling *MinDepth*. Finally, each iteration in Algorithm *FindMinCost* halves the distance between C_{\min} and C_{\max} . Noting that proxy costs cannot exceed $|P| \cdot \max_{p \in P} f(p) \cdot g(1)$ which is $O(n)$, there can be at most $O(\log n)$ iterations. ■

6.3 Heuristics for Non-Uniform-Edge Overlays

In this section, we focus on the development of a heuristic that seeks to minimize the cost of a delay-bounded, fanout-constrained multicast with maximum delay Δ in a network where end-to-end delays between nodes can vary. It can be shown that solving this problem exactly is NP-hard via a reduction to Hamiltonian path. Details of this reduction are provided in Appendix A

6.3.1 Design from Theoretical Observations

We draw two observations from our theoretical results in the previous section:

- It is beneficial to have nodes with high fanout closer to the root of the tree.
- One approach to finding the minimum cost delay-bounded tree is to construct an artificial bound on cost by limiting aggregate fanout permitted over all proxies, and then varying this bound to identify a minimum cost tree that meets the delay bound.

Obviously, our theoretical results do not account for the variation in end-to-end delays between nodes. Clearly, it is beneficial to keep delays of hops near the root of the tree small since edges near the root affect a larger fraction of receivers than do their descendants. A dilemma arises when there are a class of nodes with high permitted fanout connected to edges with high delay and another class of nodes with small permitted fanout connected to edges with small delay. Should we move nodes with high fanout nearer to the root, or edges with small delay? Our approach is to develop a heuristic with a tunable parameter, α that can be varied between 0 and 1 whose value determines the relative importance of edge delays and permitted node fanouts when making this selection.

We now describe the process used by heuristic $\text{FindTree}(\Delta, C)$ in its attempt to build a tree with delay bound less than Δ and maximum cost less than C . During the running of the heuristic, a proxy budget B is maintained that limits the set of edges that can be used from proxies. Initially, this budget is set to C , and every time a new edge is used in the tree extending from a proxy, the cost of adding that edge is deducted from B . When B becomes null the heuristic can not use proxies anymore even if there are still proxy edges available.

During its execution, the heuristic also maintains three sets of nodes:

- S_a is the set of nodes already attached to the tree and able to accept more children. Initially, $S_a = \{s\}$.
- S_o is the set of nodes to be attached. Initially, $S_o = N \cup P$.
- S_f is a set of nodes that have been attached to the tree but can no longer accept additional children because of fanout restrictions. Initially, $S_f = \emptyset$.

Throughout the duration of the running of the heuristic, S_a, S_o, S_f remain mutually exclusive sets with $S_a \cup S_o \cup S_f = \{s\} \cup N \cup P$. While $S_o \cap N \neq \emptyset$, the heuristic repeats the following procedure. For each node $n \in S_o$, the minimum distance, $\delta(s, n)$ from s to n along a path of nodes whose hop prior to n is some $n_1 \in S_a$ is computed. This ensures that were n to attach to n_1 , this would not violate the fanout constraint of n_1 . A set S is formed of the nodes $n \in S_o$ where all of the following hold:

- $\delta(s, n) \leq \Delta$

- Adding n does not incur costs that violate the remaining proxy budget, B (this is of concern when $n \in P$).

If $S = \emptyset$, then the heuristic returns a null tree (indicating failure to find an acceptable tree). Otherwise, the node $n \in S$ is selected that maximizes

$$\mathcal{H}_\alpha(n) = \alpha \frac{f(n)}{f_{\max}} + (1 - \alpha) \frac{\delta_{\min}}{\delta(s, n)}. \quad (6.4)$$

where f_{\max} and δ_{\min} are normalization constants, such that for all n , $\frac{f(n)}{f_{\max}}$ and $\frac{\delta_{\min}}{\delta(s, n)}$ lie between 0 and 1. These constants are calculated as

- $f_{\max} = \max_{n \in \{s\} \cup N \cup P} f(n)$.
- $\delta_{\min} = \min_{n \in \{s\} \cup N \cup P} d(s, n)$

where $d(s, n)$ is simply the shortest path from s to n (ignoring budgets and fanout constraints, e.g., just applying Dijkstra's algorithm).

The node n is then attached to the tree through n_1 , and is moved from S_o into S_a . Nodes n and n_1 are then moved from S_a to S_f if their respective numbers of children equal their respective fanout constraints.

Without setting a delay bound, i.e. $\Delta = \infty$, this heuristic finds the minimum depth tree that corresponds to a fixed cost (budget) C and a fixed parameter α . To find a tree with minimum cost, one method can use algorithm FindMinCost() and call FindTree() with $\Delta = \infty$ instead of calling FindBestProxyTree().

6.3.2 Additional Modifications

Evaluation with this preliminary heuristic revealed two problems:

1. The proxy budget, B would on occasion be depleted by attaching unneeded chains of high fanout proxy nodes with low delay edges that never had any end-system descendents. These proxy nodes would serve no use in the session since they would not lie on the any of the end-systems' transmission paths, but would still prevent the heuristic from subsequently adding additional proxies that might in fact serve some use.
2. Proxies would often aggressively be attached near the source, depleting available edges of nodes near the source. This would sometimes push end-systems away from the source, unnecessarily increasing transmission delays.

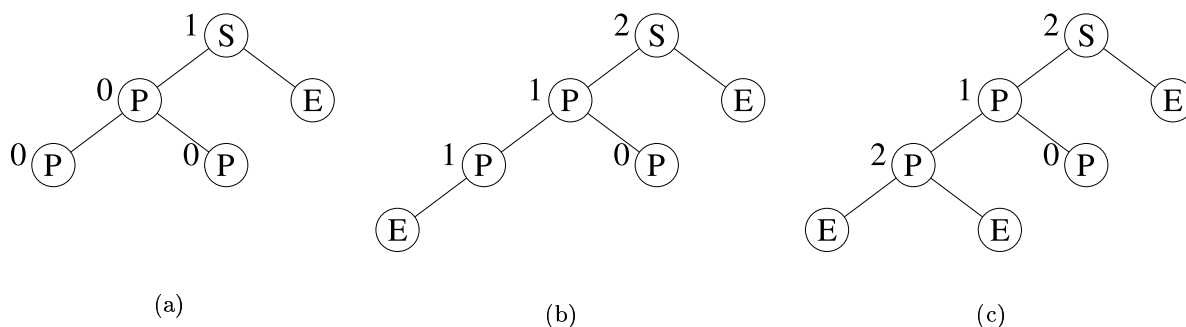


Figure 6.2: Sample 2-phase proxy attachment

These problems were rectified in a second version of the heuristic, $\text{FindTree2}(\Delta, C)$, that was identical to the first heuristic except that a 2-phase process to attach proxies was used. In this 2-phase process, nodes are attached as in the previous version of the heuristic, but an attached proxy, $p \in P$ would only be counted against proxy budget or against the fanout constraint of an ancestor node in the tree when some descendent of p (attached later on by the heuristic) is an end-system. An example of this 2-phase process is depicted in Figure 6.2. The node marked with an S is the source, nodes marked with E are end-systems and nodes marked with P are proxies. The number of “counted” children of a node is indicated to its immediate left. In Figure 6.2(a), proxies have been added but are not ancestors to any end-systems and hence do not “count” as descendents. In Figure 6.2(b), an end-system is added, incrementing each node’s child count up by 1 all the way up the chain to the source. In Figure 6.2(c), an additional end-system node is attached to the same node as before, increasing its parent node’s child count. However, that node had “counted” previously, its parent does not increment its own child count. At the completion of the running of the heuristic, children that are not “counted” do not have any end-system descendents, and hence can be dropped from the tree.

When $\alpha = 1$, the heuristic gives priority to nodes with larger fanout. Here, the heuristic most closely resembles the algorithm from Section 6.2 where edge delays are used only to break ties. Also, the heuristic is identical to the heuristic used within [30]. When $\alpha = 0$, the heuristic gives priority to nodes with minimal delay from the source. Values of α between 0 and 1 give priority to nodes with both high fanout and low delay, with the emphasis placed on fanout increasing with increasing α .

6.4 Heuristic Evaluation

In this section, we evaluate the heuristic $\text{FindTree}()$ in transit-stub networks [131] created using the *gt-itm* software package [21]. While debate continues about the accuracy of topologies generated to emulate large-scale networks [42], our understanding is that the package provides an

accurate model of the medium-sized network topologies we experiment upon within this study.

6.4.1 Experimental Setup

Underlying Network Topology

For all experiments, we generate two instances of the underlying network layer topology. Both instances contain 30 nodes within the transit domain, 10 of which are randomly selected to connect to 10 separate stub domains. Each stub domain contains 20 nodes including an *edge router* that connects the stub domain to the transit backbone. This gives a total of 30 transit nodes, 10 edge routers and 190 (non-edge) stub nodes. Edges (links) are constructed for the transit domain according to the distribution defined within the Waxman model [126] with $\alpha_W = 0.3$ and $\beta_W = 0.3$ (we use the “W” subscript to distinguish these parameters used within the Waxman model from the α parameter used within the heuristic). The two underlying network topology instances differ in how α_W and β_W are set when generating edges (links) within the stub domain. In one instance, we generate *sparse* stub domains by setting $\alpha_W = 0.3$ and $\beta_W = 0.3$. In the *dense* instance, we set $\alpha_W = 0.6$ and $\beta_W = 0.7$. The delay assigned to each edge is proportional to its length.

Overlay Generation

For each experimental run, a fixed number of end-systems are connected to randomly chosen stub nodes (nodes are chosen with replacement such that a single stub node can connect to multiple end-systems), and 10 proxy nodes are selected. We consider four ways of assigning proxies to nodes in the network:

- Proxy placement is restricted to the transit backbone.
- Proxy placement is restricted to stub nodes.
- Proxy placement is restricted to edge nodes (i.e., the bridges between transit and stub)
- Proxy placement is unrestricted, with each proxy having equal likelihood of being attached to a transit, stub or edge node.

The delay between a node and the proxy connected to that node is set to 0 (i.e., the proxy is co-located with the node). Delay from a stub node to a connecting end-system is set to 0.3 times the average delay between the source and stubs that contain end-systems. The fanout from a proxy node is chosen from a uniform distribution between 5 and 15. End-system fanout is uniformly chosen between 1 and 3.

6.4.2 Performance Evaluation

Figure 6.3 plots the cost of the tree for a session with 100 end-systems in which end-systems meet a given delay bound. The cost function used within the simulation is the sum of the fanouts of edges extending from proxies within the tree, (i.e., the cost function is $g(i) = i$). The different sub-figures plot results for the various proxy placement restrictions. In each sub-figure, the value of the x -axis is $\Delta / \max_{n \in N} d(s, n)$, i.e., the delay bound normalized to the largest end-system end-to-end (unicast) delay from the source (in the underlying network). The y -axis gives the cost. Each curve plots, for a given α , the cost of the tree computed by the heuristic to meet the delay bound for all receivers given on the x -axis. Each point plotted is the average of 98 simulation runs. The 95% confidence intervals shown are generated using seven sample points, where each sample point is the average of 14 runs.³ Values of α whose curves lie closer to the bottom and left-most edges of the graph are preferable, since these points represent lower cost trees that yield lower delays.

We see that $\alpha = 0$ is preferable to $\alpha = 1$ when proxies are restricted to either the stub or edge points but not when proxies are restricted within the transit portion of the network. Intuitively, this is because proxies in the transit portion of the network are typically closer to one another. Hence, the high fanout nodes that are first added into the tree when $\alpha = 1$ are closer together. In contrast, when proxies lie at the edges of the network, the distances (and hence delays) traversed to connect these high fanout nodes together becomes excessive. However, these observations are in some sense moot, since in all four scenarios, $\alpha = 0.6$ and $\alpha = 0.3$ produce trees whose cost is consistently lower to achieve any fixed bound on the delay. This demonstrates that the “best” trees result from giving a more equal consideration to both the transmission delay and bandwidth constraints. The heuristic used in [30] is equivalent to our heuristic with $\alpha = 1$. The point at which the curves touch the x -axis indicates the minimum delay achievable for all receivers in a session that receives no proxy support. We see that a value of $\alpha = 0.6$ or $\alpha = 0.3$ yields almost a 25% reduction in this delay in comparison to the case where $\alpha = 1$.

By comparing the plots across the three figures, we observe that proxies reduce costs for a fixed delay more significantly when placed at edges or in the transit portion of the network than in comparison to the edges. We have also explored how varying the number of end-systems and proxies impacts delay and cost. We see that increasing the number of end-systems grows costs, but at a very slow rate. Increasing the number of proxies decreases costs, but at a rate even slower than the increase in costs when additional end-systems are added to the session. We see the same trends irregardless of whether the stub networks are sparse or dense.

Next, we explore in detail how proxy costs of delay-bounded trees constructed by the heuristic vary as a function of the number of receivers in the multicast session. Figure 6.4 plots these results along two axes. In Figure 6.4(a), we plot the proxy cost as a function of the delay bound using

³Each simulation run determines the minimum delay obtainable for all end-systems given a fixed cost budget. Our averages are then computed over the set of delays achieved for each cost value. As a result, our confidence intervals lie horizontally for each cost, instead of vertically for each delay.

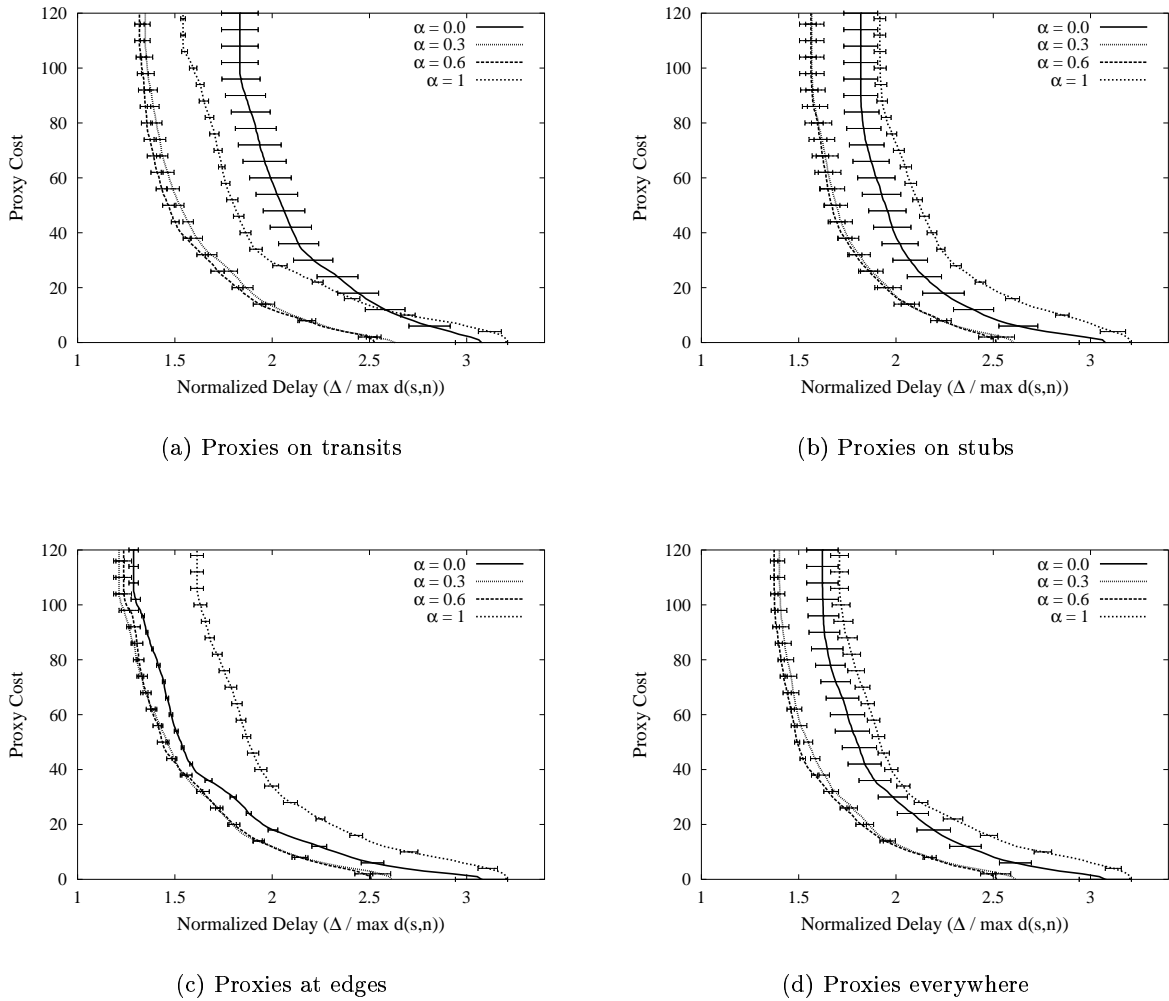


Figure 6.3: Heuristic Performance

the heuristic with $\alpha = 0.3$. Each curve plots the cost for a fixed number of receivers participating in the session. This figure demonstrates that there is an increasing cost in building a tree that meets a given delay bound as the number of receivers increases, but that this cost grows slowly. Figure 6.4(b) further illustrates this observation, plotting the cost as a function of the number of clients in the session, where each curve represents a different bound on the maximum delay to any receiver in the session. We observe an approximate linear growth in the cost as the number of clients is increased. However, we note that because it would appear as though the curves would cross the line above the y -axis, we suspect that the cost per receiver decreases as the number of receivers increase, and that the cost approaches some asymptotic value as the number of receivers grows large. In addition, the cost per receiver increases at a much higher rate as the desired delay bound is tightened. Hence, proxy-assisted multicast costs less per client as the number of clients in a session increases and as desired delay bounds are relaxed.

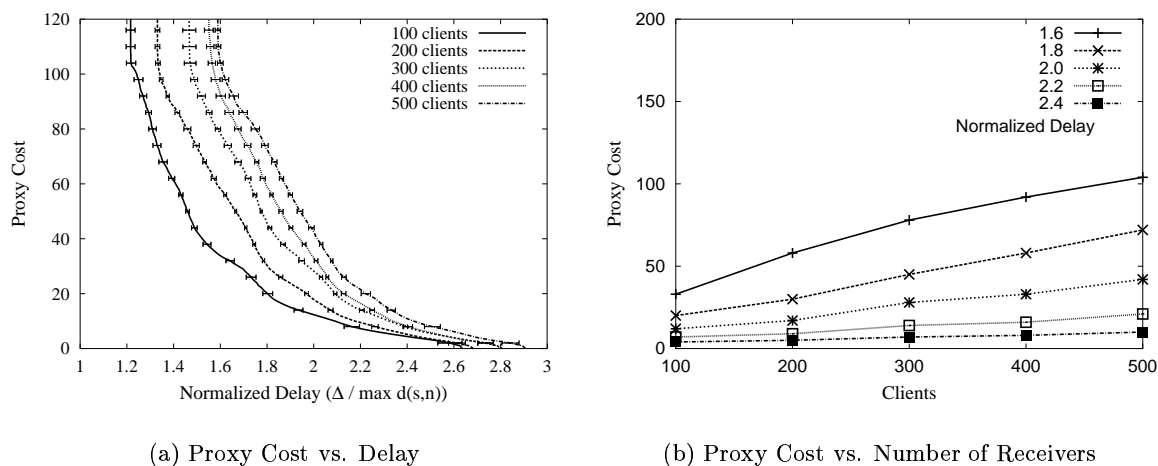


Figure 6.4: Proxy costs as a function of number of receivers and desired delay bound

6.5 Conclusion

In this chapter, we evaluated algorithms and heuristics to minimize the cost of employing proxies as a means of meeting delay constraints for hybrid end-system/proxy application layer multicast sessions. We present an algorithm that provably minimizes costs for the case where delays between application layer multicast points is uniform. For the case where delays between these points vary, solving the problem optimally is NP-hard. We instead resort to a tunable heuristic, and demonstrate that certain tunings outperform previously developed heuristics. The results demonstrate that these *hybrid* approaches are a promising means for enabling low-cost, delay bounded multicast services upon a unicast-only network layer.

Appendix A The Need for Heuristics

In Section 6.3, we claim that the problem of finding a tree whose longest path from a given source node s to any node is minimized, with the restriction that within the tree, each node n has a bound on its fanout, $f(n)$, is NP-hard. There is no obvious reduction from a Steiner Tree problem. Also, we seek a shortest paths tree, and not a minimum spanning tree as in [113, 12, 11]. We provide a sketch here of the reduction from Hamiltonian Path. While we doubt the results presented here are novel, we include them for completeness.

We begin by considering a graph $G = (N, E)$ that is not fully connected in which all edges have unit length. If we set $f(n) = 1$ for all $n \in G$ (bounding the permitted fanout that can be used to construct the tree, not bounding its fanout in the underlying graph, G), then a shortest-path tree would have depth $|N| - 1$ iff a Hamiltonian Path existed within the graph, and finding a Hamiltonian path is known to be NP-complete [54]. Thus, finding a shortest-path tree in G

is NP-hard. To show the problem is NP-hard for graphs in which the tree can have arbitrary (but bounded) fanout $f(n) < N$, we construct a new graph G' that is identical to G except we add some additional nodes and edges. For each node n with $f(n) > 1$, create $f_1(n) = f(n) - 1$ additional nodes and create edges in G' that attach these new nodes to (and only to) n . Finding a shortest-path tree in G' that includes all nodes in G' will require n to attach to the new $f(n) - 1$ in the tree that only it attaches to in the underlying graph G' . If a tree can be constructed that does not violate fanout constraints in G' , then we have constructed a Hamiltonian Path for the arbitrary graph, G . Thus, the problem of constructing a minimum-depth tree with an arbitrary FCF is NP-hard.

Last, to show that the above problem is NP-hard in a fully-connected graph G with variable-length edges, we construct G as follows: Choose an arbitrary graph with unit edge-lengths of 1. Wherever an edge does not exist, add an edge of length $\Delta + 1$, where Δ is the desired delay bound. Clearly, these new edges cannot be used in a tree that solves our problem, so solving the problem reduces to solving the problem on an arbitrary graph with unit length edges. Thus, solving the problem on a variable-length edge fully-connected graph is NP-hard.

Chapter 7

Bandwidth-Sharing Schemes in End-System Multicast

Multi-party applications such as distributed gaming, tele-video conferencing, and distributed concerts require simultaneous transmission from a set of source points to possibly overlapping sets of receiving points. For instance, in a distributed game, to keep all players' perspectives consistent, each player must update their position information and status to other players whose virtual positions are located close by [55]. Tele-video conferencing and distributed concerts require that the participants can almost simultaneously see and hear one another. Hence, it is important to configure these sessions such that transmissions are received within a given delay bound. However, multicast sessions also share the same overlay resources and specific mechanisms must be designed to optimize the usage of these resources and especially the bandwidth.

While there have been several recent works that develop efficient algorithms for building overlays for group communication, there are few work that has explored how to construct session topologies in environments where multiple sessions, whose memberships vary dynamically with time, compete for the same network resources.

Previous work in IP multicast that considered multiple sessions competing for resources is in the context of congestion control [15, 95, 127, 108]. There, sessions are expected to reduce bandwidth requirements when insufficient bandwidth is available to avoid proliferating a congested state within the network.

In this chapter, we evaluate overlay construction algorithms that construct overlay trees in environments where each node can receive and forward data from multiple senders in the same network. Our delay optimization criteria is to build depth-bounded trees that keep the number of *hops* through the overlay that are taken to reach a receiver within a certain bound. An algorithm that minimizes the number of hops does not necessarily minimize the end-to-end propagation delay to a receiver as the delays between different pairs of overlay nodes can differ. Nonetheless, there are several reasons why hop-depth bounded trees are of interest. It is a non-trivial task to obtain accurate estimates of propagation delays between overlay participants. Furthermore, delays that occur in transmission between such end-systems is often dominated by a combination

of the delay across last-mile technology such that DSL, modem or cable and the lack of priority given to processing of arriving and departing packet transmissions. Moreover, some overlay multicast protocols, e.g. [73] use the bandwidth as the primary routing metric.

The key issue that arises in the context of multiple senders is how to allocate a node's forwarding capacity in the tree that is currently being constructed or modified, given that 1) participation in the tree may vary, forcing changes in the topology and 2) other sessions might subsequently be initiated that require access to these same overlay resources. To address this issue, we consider two baseline algorithms, CLUSTER and DISPERSE (Section 7.2), whose approaches toward selecting the nodes that support transmission for a session lie at two extremes. CLUSTER attempts to apply a node's outgoing bandwidth toward a single session, thereby *minimizing* the number of sessions for which a node acts as a forwarding agent. In contrast, DISPERSE splits a node's outgoing bandwidth across as many sessions as possible while maintaining the depth-bound in the constructed tree, thereby *maximizing* the number of sessions for which a node acts as a forwarding agent.

We begin by showing via a mathematical analysis that CLUSTER is optimal in homogeneous environments where all nodes have the same bandwidth limitations and all multicast sessions have the same set of requirements (Section 7.3). We then turn our attention to heterogeneous networking environments where nodes have variable capacity, and overlay participants can join and leave the network. We begin by modeling the heterogeneous environment as a queueing system and use this system to derive a theoretical lower bound on the blocking probability of algorithms (assuming Poisson arrival of sessions) in these settings (Section 7.4). This lower bound translates into an upper bound on the rate at which sessions can be admitted into the network. Since no algorithm can perform better than this theoretical bound, it provides an additional benchmark to evaluate algorithm performance. Here, we show near optimality by demonstrating that, by merely allowing the sender of an arriving session to transfer its current transmission responsibilities for other sessions to other nodes, CLUSTER and DISPERSE approach the theoretical bound (Section 7.5). Last, we use simulation to evaluate these algorithms in settings where session membership varies with time (Section 7.6).

7.1 Network Model

In this section, we present the network and session models that will be used to evaluate our algorithms that form session trees for multiple sessions. Let \mathcal{N} be the set of receivers, i.e., the set of nodes that wish to participate in at least one multicast overlay session. We define $\mathcal{S} \subset \mathcal{N}$ to be a sequence of these nodes that wish to act as transmitters for a session. Note that this set can contain repetitions, i.e., a node appears k times in \mathcal{S} if it is the sender for k different sessions. We denote the i th entry in the sequence by s_i .

Let $\mathcal{R}_i \subset \mathcal{N}$ be a set of nodes that wish to receive the transmissions from sender s_i . By convention, we require that $s_i \in \mathcal{R}_i$. We assume that a node n is only willing to participate in

overlay multicast session when it is interested in receiving the transmitted data of that session. In other words, \mathcal{R}_i is exactly the set of nodes that form the multicast tree that will carry data for the session for which s is the sender. Note that there will be a set of leaf nodes in this multicast tree that will be receiving, but will not be forwarding these transmissions.

The i th session forwards data to all participants at an arbitrary rate, ρ_{s_i} . β_n is the aggregate outgoing bandwidth of node n , such that if n forwards directly to c_j nodes for session j , then it must be the case that $\beta_n \geq \sum_{j \in A} c_j \rho_{s_j}$ where A is the set of active sessions. This requirement is applicable to several last-mile technologies such as cable and DSL that provide asymmetric bandwidths in the two directions, where incoming bandwidth is significantly larger (and essentially unbounded) in comparison to the outgoing bandwidth.

We define Δ_i to be a bound on the depth of the tree for the session whose transmissions emanate from s_i , i.e., no node should be more than Δ_i overlay hops from the source node. For simplicity of presentation, we assume that $\Delta = \Delta_i$ is constant for all sessions.

We design algorithms that operate in two settings. In the static setting, the set of participating sessions are known prior to the establishment of the overlay. In the dynamic setting, we permit session membership to vary with time, i.e., nodes join and leave individual sessions. Such changes in session membership are common in multicast applications and often necessitate reconfiguration of overlays as the original overlay can become partitioned when nodes in the middle of the overlay tree leave the session.

7.2 Algorithms

We now describe the algorithms that we use to determine whether or not to admit an incoming session, and, when admitted, the topology structure of the tree.

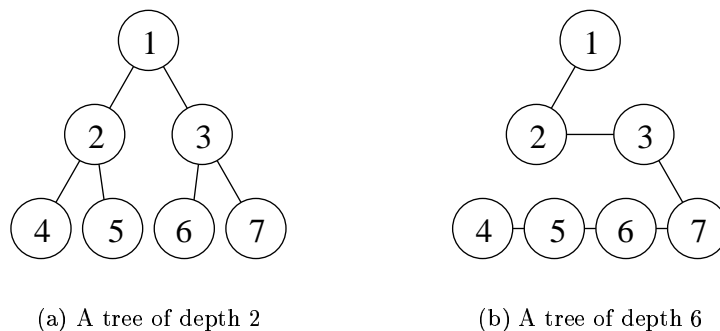


Figure 7.1: Selecting “good” clusterings

To give the reader a feel for what makes a “good” algorithm, consider the trees constructed in Figure 7.1. Assume that each node in this figure is capable of providing direct transmissions to two additional nodes. In Figure 7.1(a), 3 nodes have been selected, each of which is fully utilizing its outgoing transmissions to connect all nodes within a tree in which node 1 is the source. The

tree formed has depth 2, and, in addition, another tree of depth 2 can be constructed (e.g., node 4 transmits to nodes 5 and 6, node 5 transmits to nodes 2 and 3, node 6 transmits to nodes 7 and 1). Hence, trees that cluster nodes within a session are intuitively “good”. In contrast, the tree in Figure 7.1(b) has depth 6 and precludes any other trees of depth less than 6 from being constructed.

This figure demonstrates that intuitively, it is preferable to utilize the outgoing edges of a node within a single tree, rather than to spread these edges among several trees. We shall see, however, that this intuition can be misleading in dynamic and heterogeneous environments.

7.2.1 Algorithm CLUSTER

CLUSTER forms trees by utilizing a set of nodes for forwarding such that the nodes’ remaining available bandwidth is minimized. The algorithm distinguishes between *partial nodes*: nodes whose bandwidth has already been applied within other sessions, and *full nodes*: nodes that do not currently forward transmissions on behalf of other sessions. CLUSTER tries to incorporate as many already-existing partial nodes as possible into the middle (i.e., not as leaves) in the tree before incorporating full nodes into the tree.

Algorithm CLUSTER(s_i):

1. Let P be the sequence of partial nodes in \mathcal{R}_i that are sorted in the decreasing order of available bandwidth capacity. Let U be the set of full nodes in \mathcal{R}_i , sorted by decreasing available bandwidth capacity.
2. Determine the minimum m for which a tree T can be constructed connecting all nodes in \mathcal{R}_i where m nodes in U are non-leaf nodes and a possibly empty subset of nodes $P' \subseteq P$ as non-leaf nodes in which
 - $D_{\max}(T) \leq \Delta$
 - nodes with higher available bandwidth capacity appear at a lesser depth (closer to the source).
 - any set of partial nodes P'' that satisfies $P' \subset P'' \subseteq P$ cannot be used as non-leaf nodes in a tree T' that satisfies $D_{\max}(T') \leq \Delta$.
3. If such an m exists, build the tree from m nodes drawn from U and the remaining nodes drawn from the set P' . Otherwise, return that no such tree can be built.

We note that determining the minimum m for a given set P' can be done in time $O(|\mathcal{N}| \log |\mathcal{N}|)$. This follows from Lemma 6.1 in the previous chapter that proves that the minimum depth tree is the one where nodes with greater bandwidth availability are placed closer to the source of the tree. Hence, it is sufficient to build the tree by first sorting nodes in order of decreasing fanout, and then attaching the nodes to the tree in this order to the node of minimal depth that contains available edges.

7.2.2 Algorithm DISPERSE

Algorithm DISPERSE forms trees by utilizing as many nodes as possible for forwarding without exceeding the depth constraint. Barring a depth constraint, a chain of nodes is the preferred tree for DISPERSE.

Algorithm DISPERSE(s_i):

1. Let S be the sequence of nodes in \mathcal{R}_i sorted in order of decreasing available bandwidth capacity. Let c be the number of children that the first member of S can support for the session.
2. Let $d_{\min}(c)$ be the depth of a minimum-depth tree, $T(c)$ that can be built using no more than bandwidth c from any node in S . We use the majorization method introduced in the previous chapter to form this tree in time $O(|\mathcal{R}_i| \log |\mathcal{R}_i|)$. If $d_{\min}(c) > \Delta$, return that a tree cannot be constructed.
3. Form the tree $T(c - 1)$ with depth $d_{\min}(c - 1)$. If $d_{\min}(c - 1) > \Delta$, return tree $T(c)$. Otherwise, decrement c and go to step 2.

The reason why we use an iterative algorithm is that with a first non-iterative version of the algorithm, we noticed that when the delay constraint is not **very** large, then the performance of DISPERSE retrogrades drastically in comparison to CLUSTER. This is because DISPERSE tends to build chains. With the actual version, we should notice that DISPERSE do more than “dispersing” the bandwidth but it tries to find the adequate fanout that should be used from each node to satisfy the depth constraint.

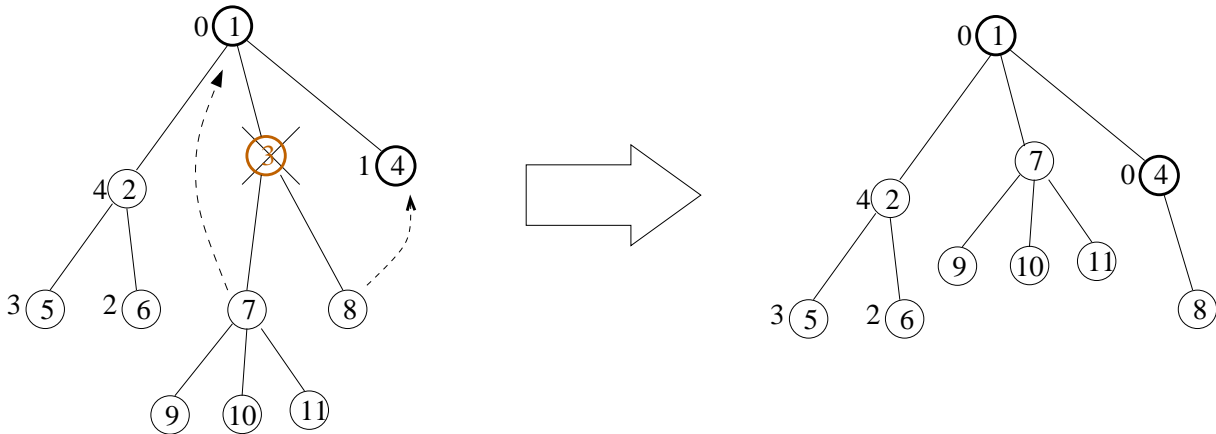
7.2.3 Algorithm Extensions

The algorithms described above have two shortcomings. Here, we describe these shortcomings and present simple extensions to our algorithms that address these shortcomings.

The first shortcoming addresses the utilization of the node that will transmit data. This node may not have sufficient bandwidth to forward its own transmissions because its bandwidth is being utilized to forward transmissions of other sessions. We consider three possible actions that we refer to as *variants* that the network can take when faced with such a situation.

- no-swap: If s_i does not have enough outgoing capacity to initiate a session, the session is rejected.
- swap: s_i shifts some of its forwarding responsibility to another node that is participating in the same sessions as s_i that has the bandwidth capacity to take on the additional bandwidth load. If no other node can be located that has sufficient spare capacity, the session to originate from s_i is dropped.

- reservation: A fixed amount of outgoing capacity is reserved at every node specifically for the transmission of a session that is initiated at that node. At other times, the bandwidth must remain unused.



The subtree of greatest depth is attached at the highest node: node 7 is attached to the root of the tree at height 0. Then node 8 is attached to a node at height 1. CLUSTER selects node 4 with the minimum available bandwidth.

Figure 7.2: The leave procedure

The second shortcoming addresses the *dynamics* of membership within sessions. In particular, it is likely that there will be applications in which session members arrive late and leave early. Late arrivers must be fit into the tree within the depth bounds. By leaving, a node that forwards transmissions for the sessions will disconnect the tree. This disconnection must be repaired.

Our attachment and disconnect procedures for the two algorithms attempt to mimic the traits exhibited by the original algorithm. For CLUSTER, a late joiner is attached to the node with minimum available bandwidth that is sufficient to support a late joiner such that the added node lies within the depth bound. If no such node exists, then the late joiner is attached to a node beyond the depth bound that has sufficient bandwidth to support such a node, when one exists. The node is otherwise rejected. The reason why we proceed to attach nodes in situations where the depth bound is violated will be explained below. When a node leaves the session, its children become the roots of detached subtrees, where the subtrees can have variable depth. These subtrees are then attached at the highest points at which there is available capacity to perform the attachment, with the subtrees of greatest depth being attached at the highest point. When there are several nodes at the same height to which an attachment can be made, the node with minimum available bandwidth is selected (Figure 7.2). A subtree is discarded when there is insufficient available bandwidth to connect the root of the subtree.

For DISPERSE, a late joiner is attached to the node with maximum available bandwidth that lies within the depth bound. If no such node exists, then a node with maximum available bandwidth that lies outside the depth bound is used. If no node has sufficient available bandwidth to forward transmissions to the late joiner, the late joiner is rejected. The algorithm to re-attach detached subtrees after a node leaves the session is the same as that for CLUSTER, except that the node with maximum available bandwidth is selected when there are several nodes at the same height to which an attachment can be made (in Figure 7.2, node 8 is attached to node 2 instead of node 4).

The tree formed as a result of nodes joining and leaving the session may be able to admit fewer nodes into the session beyond the depth bound than one constructed in a static setting. For this reason, the tree is periodically reconfigured by applying the CLUSTER algorithm or the DISPERSE algorithm to the set of nodes that are currently participating in the session. Hence, nodes that are attached beyond the depth bound can be brought within the depth bound during this reconfiguration. Those nodes that lie outside the depth bound after a reconfiguration are dropped from the session.

7.3 Homogeneous Network

In this section, we present a formal proof that algorithm CLUSTER with swap is optimal within the homogeneous setting except for a small number of session configurations.¹ By homogeneous, we mean that $\mathcal{R}_s = \mathcal{N}$ for all $s \in \mathcal{S}$, where $\rho_s = \rho_{s'}$ for all $s, s' \in \mathcal{S}$, and where $\beta_n = \beta_{n'}$ for all pairs $n, n' \in \mathcal{N}$. We set $F = \lfloor \beta_n / \rho_s \rfloor$, the number of session copies that each node can forward, and let Δ_{\min} be the minimum depth over all trees that can be constructed within the available bandwidth constraints that includes all receivers. We use the following notation introduced in the previous chapter (Section 6.1): $D_n(T)$: is the depth of node n in tree T where $D_s(T) = 0$ for the source of the tree s and $D_n(T) \equiv D_{\Pi_n(T)}(T) + 1$, where $\Pi_n(T)$ is the parent in the tree T of node n . $D_{\max}(T, S) = \max_{n \in S} D_n(T)$, i.e., the maximum depth in T of any nodes in $S \subseteq T$. We also define $D_{\max}(T) \equiv D_{\max}(T, T)$.

We let T_k equal the k th tree that is constructed with source node s_k by Algorithm CLUSTER. We define the FCF, $f_k(n)$, to equal the number of edges that connect from node n to its children in tree T_k . During the running of the algorithm, we define the k th *configuration* of sessions, C_k to be the set containing the first k trees $\{T_1, \dots, T_k\}$ to be constructed, with C_0 being the configuration containing no trees. We define $f_{\Omega(C_k)}(n)$ to be the number of *residual* or as-of-yet unused edges of a node n within configuration C_k (i.e., after the first k trees have been constructed). Because our setting is homogeneous, we have that $f_{\Omega(C_0)}(n) = f_{\max}$ for all $n \in \mathcal{N}$. We define a partial node $p \in P$ to be *usable* under configuration C_k whenever $f_{\Omega(C_k)}(p) < F$ and it is possible to build a tree T' where $D_{\max}(T') \leq \Delta$ using node p and along with other nodes n

¹For this small set, we conjecture that CLUSTER is optimal, but we have not yet formally proven this conjecture.

that satisfy $f_{\Omega(C_k)}(n) = f_{\max}$. In other words, it is feasible to use p to build tree T_{k+1} .

Lemma 7.1 *Under Algorithm CLUSTER, tree T_1 contains at most one non-leaf node n_0 for which $f_1(n_0) < F$. Letting m equal the number of non-leaf nodes in tree T_1 that satisfy $f_1(n) = F$, then remaining trees, T_k contain either m or $m + 1$ non-leaf nodes, where at most two nodes, n_1, n_2 exist for which $f_k(n_j) < F, j = 1, 2$. Last, under any configuration C_k , there is at most one usable node in P .*

Proof: The fact that T_1 contains at most one node n_0 for which $f_1(n_0) < F$ follows trivially from algorithm CLUSTER, given that all nodes initially have $f_{\Omega(C_0)}(n) = f_{\max}$, and CLUSTER turns at most one node into a partially utilized node for each tree that it builds. We show the 2-node partially utilized property for the remaining trees by induction. Assume the result holds true after building k trees, $k \geq 1$. The algorithm first uses any usable nodes in P when constructing the $k + 1$ st tree. Furthermore, the algorithm does not introduce an additional non-leaf node $n \in U$ until all nodes already present in the tree have all edges utilized. Hence, in the end, at most one non-leaf node in the tree is partially utilized, completing the inductive step.

Last, we note that when no usable nodes are available in P in configuration C_k , tree T_{k+1} will be isomorphic to T_1 . If a usable node p exists in P within configuration C_k , the tree must still contain at least $m - 1$ nodes n for which $f_{k+1}(n) = D_{\max}(k + 1)$ (or else T_1 could have been constructed using fewer than m nodes that satisfy this property). If these $m - 1$ nodes and p do not produce a sufficient number of edges to connect the remaining (leaf) nodes, then an m th node is brought in. If this is not sufficient, then an $m + 1$ st node is brought in. This will clearly be sufficient since m of these $m + 1$ nodes satisfy $f_k(n) = F$ and hence could be used to build a tree isomorphic to T_1 . Node p and the last node to which children are assigned in are the only two nodes that have children in T_{k+1} that do not apply all F edges within this tree. ■

Lemma 7.2 *If $\Delta > \Delta_{\min}$, then Algorithm CLUSTER(s) builds the maximum number of trees possible with depth no greater than Δ .*

Proof: We begin by noting that since all senders transmit at the same rate, if a node's remaining available bandwidth is insufficient to be a non-leaf node in tree T_i , it will also be insufficient for all trees T_j for $j > i$. We prove inductively on i that Algorithm CLUSTER builds T_i of depth at most $\Delta_{\min} + 1$, that only one node remains in P after the building of each tree, and that this one node is usable. The inductive step holds for T_1 : straightforward application of Lemma 7.1 gives us that T_1 builds a tree of depth Δ_{\min} . Furthermore, it is clear from the design of CLUSTER that in T_1 , there is at most one node added to the tree for which only some of its available edges are used when building is complete. Hence, in configuration C_1 , there is at most one node in P .

To continue our proof by induction, we first note that a node $p \in P$ within configuration C_k is always usable in the building of tree T_{k+1} . This can be seen by considering the worst case,

where p has only one available edge to support downstream nodes, i.e., $f_{\Omega(C_{k-1})}(p) = 1$. Here, it is possible to build T_k within the depth constraint by having the source send to p , and then extend the rest of the tree from p . This subtree can be built isomorphic to T_1 with one leaf node removed (since one node p has already been added). Since T_1 has depth Δ_{\min} , the T_k described above has depth $\Delta_{\min} + 1$. For the case where p has more than one usable edge, these additional edges could be used to further decrease the depth of T_k will not increase. In addition, since CLUSTER will arrange these nodes to minimize the depth (seen by applying Lemma 7.1), the T_k built by CLUSTER will have depth less than Δ .

The Lemma is then proved by considering the FCF of remaining edges, $f_{\Omega(C_k)}()$ after the algorithm has generated k trees, and let $g()$ represent the fanout function of remaining edges of other nodes from any other arbitrary construction of k trees. Since each tree utilizes $n - 1$ edges, the total number of edges remaining is the same in the two cases, i.e., $\sum_{n \in \mathcal{N}} f_{\Omega(C_k)}(n) = \sum_{n \in \mathcal{N}} f_1(n)$. $f_{\Omega(C_k)}() \succ f()$ follows immediately from the fact that $f_{\Omega(C_k)}()$ contains at most one node in P . ■

Lemma 7.3 *If $\Delta = \Delta_{\min}$ and $|N| \leq X - F$ or $|N| \geq X - F/2$, where $X = (F^{\Delta+1} - 1)/(F - 1)$, then Algorithm CLUSTER(s) builds the maximum number of trees possible with depth no greater than Δ .*

Proof: We prove this result by considering two cases. First, consider the case where $|N| \geq X - F/2$. In each tree T_k generated by CLUSTER, all but one node at depth less than Δ_{\min} have F children, and the other has $X - |N| \geq F/2$ children. If $X - |N| = F/2$, then P is empty during even configurations, and during odd configurations contains a single node p where $f_{\Omega(C_k)}(p) = F/2$, and is therefore usable (and used in building the next tree). If $X - |N| > F/2$, then CLUSTER leaves k nodes in P . However, any algorithm must add at least one additional node p in P after each tree is built where $f_{\Omega(C_k)}(p) = F - X + |N|$. This follows from a pigeon-holing argument that shows that no node in a tree T_k can contribute fewer than $X - |N|$ edges. Thus, any node that contributes some but not all edges must contribute at least $X - |N| > F/2$ edges in one tree, and therefore it does not have the requisite number of edges to contribute to a subsequent tree. Since no partial nodes can be used from previous iterations, at least $F - X + |N|$ edges remain unused (and unusable in future trees) after the building of each tree. Since this is the number of unusable edges generated by CLUSTER, CLUSTER minimizes the number of unusable edges and hence maximizes the number of trees that can be built.

For the second case, where $|N| \leq X - F$, the argument is similar to that used to prove Lemma 7.2 by showing that there is always at most one node $p \in P$ during any configuration C_{k-1} and that this node can always be used to build tree T_k with depth no more than Δ . This is because one can construct a tree entirely from unused nodes where some node at depth $\Delta_{\min} - 1$ has no children. This node can be replaced by the node $p \in P$ and $f_{\Omega(C_{k-1})}(p)$ leaves can be assigned from some other node n to p . By doing so, p is removed from P in C_k and n is inserted into P

(and is the only node in P). The shifting of leaves to p does not increase the depth of T_k , hence its depth remains within Δ . ■

The following theorem follows trivially via application of Lemmas 7.2 and 7.3.

Theorem 7.1 *Algorithm CLUSTER(s) maximizes the number of trees that can be formed from nodes in \mathcal{N} whenever $\Delta > \Delta_{\min}$ or when $|N| \leq X - F$ or $|N| \geq X - F/2$, where $X = (F^{\Delta+1} - 1)/(F - 1)$.*

Corollary 7.1 *Assume that $\Delta > \Delta_{\min}$ or $|N| \leq X - F$ or $|N| \geq X - F/2$, where $X = (F^{\Delta+1} - 1)/(F - 1)$. Algorithm CLUSTER(S) maximizes the number of sessions that can be simultaneously active in a dynamic setting where session initiation and termination times occur in some arbitrary order.*

Proof: For cases where $\Delta_{\min} > \Delta$ or $|N| < X - F$, our previous results showed that CLUSTER will form a tree within the depth bound from a set of nodes in which all edges are free and one node is partially used. Since CLUSTER attempts to use as many partial nodes as possible, it “saves” unutilized nodes for when they are absolutely needed. Since each individual partially used node appeared previously in a tree, it can always do so again.

For the case where $|N| > X - F/2$, each partially used node is used only in a single tree. When the session ends and the nodes are returned to the unused pool, all edges in all returned nodes are therefore available for future use.

Detailed proof of the result can be carried out using the above arguments together with the interchange argument (see for example Liu et al. [82] for a comprehensive treatment on this technique). ■

We conjecture that the above Theorem and Corollary are still valid when $X - F < |N| < X - F/2$. However, we are unable to formally prove this conjecture at this time.

7.4 Queuing Model: A Stochastic Knapsack

We derive performance bounds on our algorithms by considering equivalent queuing systems. We map our system into a stochastic knapsack framework, and compute lower bounds on blocking probabilities based on the equivalent system.

Let \mathcal{M} be the set of all possible bandwidth requirements of Sessions in the system (i.e. product of transmission rate and session size-1). Let M be $|\mathcal{M}|$.

The stochastic knapsack consists of C resource units to which objects from M classes arrive. Objects from class m are distinguished by their arrival rate, λ_m , mean holding time, $1/\mu_m$ and their size b_m .

Let n_m denote the number of class- m objects in the knapsack. Then the total amount of resource utilized by the objects in the knapsack is $\mathbf{b} \cdot \mathbf{n}$, where $\mathbf{b} := (b_1, b_2, \dots, b_M)$ and $\mathbf{n} := (n_1, n_2, \dots, n_M)$. We define the process in terms of the state space of the different class- m objects, i.e. let

$$\mathcal{K} := \{\mathbf{n} \in \mathcal{I}^m : \mathbf{b} \cdot \mathbf{n} \leq C\}$$

The knapsack always admits an arriving object when there is sufficient room. More specifically, it admits an arriving class- m object if $b_m \leq C - \mathbf{b} \cdot \mathbf{n}$. Let \mathcal{K}_m be the subset of such states, i.e.

$$\mathcal{K}_m := \{\mathbf{n} \in \mathcal{K} : \mathbf{b} \cdot \mathbf{n} \leq C - b_m\}$$

The blocking probability B_m for a class- m call under Poisson arrival assumption is then given by [109]

$$B_m = 1 - \frac{\sum_{\mathbf{n} \in \mathcal{K}_m} \prod_{j=1}^M \rho_j^{n_j} / n_j!}{\sum_{\mathbf{n} \in \mathcal{K}} \prod_{j=1}^M \rho_j^{n_j} / n_j!} \quad (7.1)$$

As a parsimonious metric to compare performance, we look at the weighted blocking probability w_b for each Stochastic Knapsack, where we weigh the blocking probability for a class- m call with the resource request of the call b_m , i.e. if $\mathbf{B} := (B_1, B_2, \dots, B_M)$, then

$$w_b = \frac{\mathbf{B} \cdot \mathbf{b}}{\sum_{m=1}^M B_m}$$

To compute the upper bound on the number of admitted sessions in our system (which correspondingly yields a lower bound on the blocking probability), we ignore the delay constraints and map the capacity C of the knapsack into $\sum_{s \in \mathcal{S}} \beta_s$, i.e. the aggregated bandwidth available in the set of nodes wishing to transmit in the sessions. Thus, whenever there is available capacity in the knapsack (available bandwidth in the nodes), we'll assume the session is admitted irrespective of the delay constraints. The size of the object b_m is then simply the product of the size of the session (minus one) and the sending rate of the session. We also assume that the bandwidth requirements b_m are integers.

7.5 Simulation Results with Static Group Membership

In this section, we compare the performance of the algorithms proposed in Section 7.2 via an evaluation of their blocking probability, i.e., the fraction of sessions that must be turned away because the participating receivers do not have sufficient bandwidth capabilities to support the arriving session. To perform this evaluation, we use simulation ². The number of nodes, $|\mathcal{N}|$ equals 100 in all simulation runs. Sessions arrive at random points at time, where the arrival times are described by a Poisson process with rate $\lambda = 1$. These sessions last for a time that is exponentially distributed with rate ρ , which we vary over the simulation runs. For each arriving

²We have developed a discrete event simulator consisted of a set of C programs and Shell scripts

session we randomly select one node to be the source. Here, we restrict each node to being the source of at most one session at any instant in time.

We now describe the various experiments individually and discuss the conclusions that we draw from each set of experiments. In all figures, we vary ρ , which equals the expected lifetime of a session, along the x -axis. The various curves plot the weighted blocking probabilities for all variants of the two algorithms. The weights is set proportional to the rate of the session, such that the penalty for dropping a session is proportional to that session's rate. For the case where all sessions are offered at the same rate, this weighted blocking probability simply reduces to the straightforward blocking probability. We also plot the theoretical lower bound on the weighted blocking probability computed from the Stochastic Knapsack framework.

7.5.1 Experiment 1: Homogeneous case

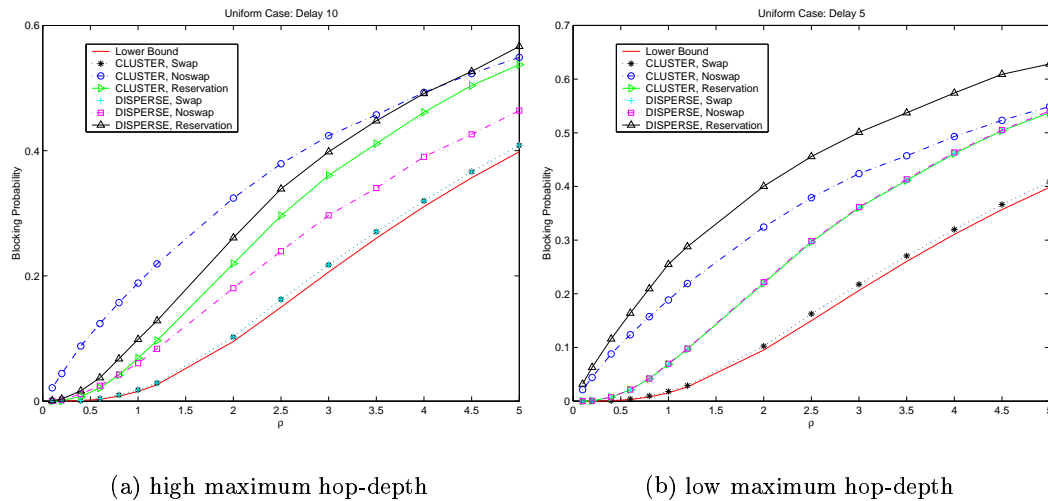


Figure 7.3: Homogeneous Setting

In this set of experiments, all receivers participate in all sessions. All session rates are fixed at 1 unit, and node capacities are fixed at 4 units. Since each session requires a set of 99 transmissions to reach all participants, a session consumes an aggregate bandwidth of 99 units. We plot results from three different sets of experiments, with different maximum hop-depth constraints. Figure 7.3(a) a medium maximum hop-depth constraint of 10, and Figure 7.3(b) depicts the results of an experiment where the maximum hop-depth constraint is a very tight 5.

From these experiments, we see that when there is a loose maximum hop-depth constraint, CLUSTER and DISPERSE with swap produce blocking probabilities that are close to the optimal, and that DISPERSE without swap has a significantly lower blocking probability than CLUSTER without swap. With a very tight maximum hop-depth constraint of 5, we observe that the blocking probability of CLUSTER with swap is close to the optimal, whereas the block-

ing probability of DISPERSE is unaffected by implementing swapping. CLUSTER without swap has a higher blocking probability than these other three. For both the algorithms and for all maximum hop-depths, reservation yields the highest blocking probability. A succinct description of the conclusions from these experiments is that for a homogeneous session configuration, CLUSTER should be used if swapping is permitted. Otherwise, DISPERSE should be the algorithm of choice.

7.5.2 Experiment 2: Variable Node Capacities

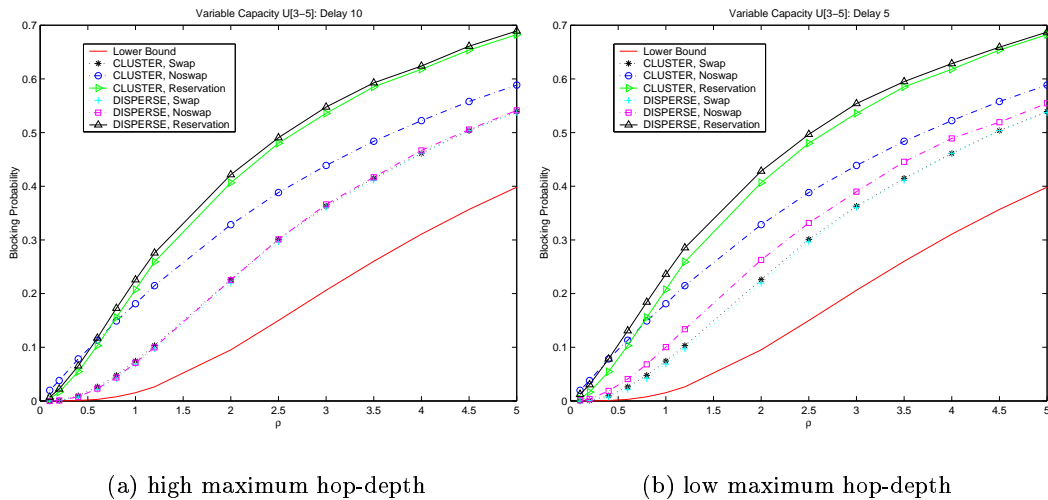


Figure 7.4: Heterogeneous setting, variable node capacity

In this set of simulations, the bandwidth capacity units of each node is selected uniformly at random from the set $\{3, 4, 5\}$. Session size is fixed at 100 and each session's rate is fixed at 1. Our conclusions from an examination of the plots in Figure 7.4 are for the most part similar to those reached in the case of a homogeneous network setting. Reservation again yields a higher blocking probability than the other variants. If swapping is enabled, CLUSTER and DISPERSE yield similar blocking probabilities. The one exception is the case where the maximum hop-depth is low. Here, using CLUSTER results in marginal improvements in blocking probability in comparison to DISPERSE. Again, the conclusion to be drawn is that if swapping is not permitted, DISPERSE is algorithm of choice. Otherwise, the choice of CLUSTER or DISPERSE is arbitrary since the blocking probabilities of the two algorithms are virtually identical. The only difference is that the simulated blocking probabilities are significantly higher than the computed theoretical lower bound on the blocking probability.

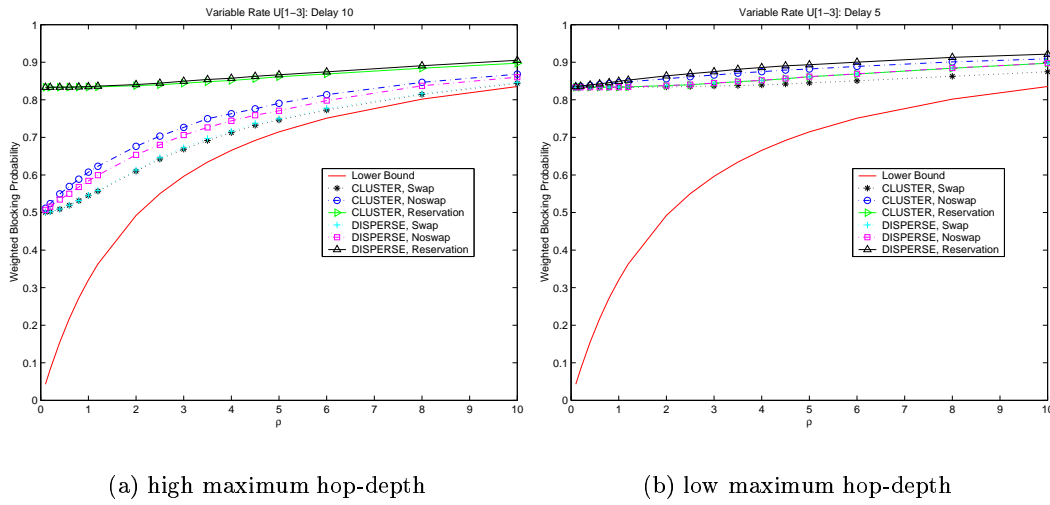


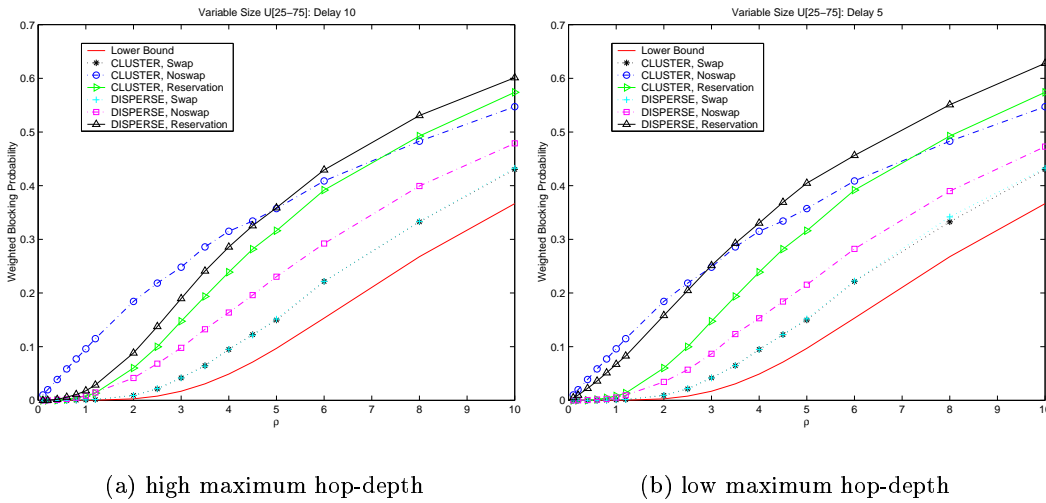
Figure 7.5: Heterogeneous setting, variable session rate

7.5.3 Experiment 3: Variable Session Rates

In this set of simulations, the rates of arriving sessions are selected uniformly at random from the set of integers ranging from 1 to 3. Node capacity is fixed at 4 and session size is fixed at 100. Examination of the plots in Figure 7.5 reveals that The weighted blocking probability that results from using DISPERSE meets the theoretical lower bound for large (unbounded) maximum hop-depths. For tighter maximum hop-depths, CLUSTER with swapping produces a slightly lower weighted blocking probability than DISPERSE, but the weighted blocking probability without swapping is significantly higher. Here, we conclude that when session depths can be unbounded, or when swapping is not permitted, DISPERSE should be used. Otherwise, CLUSTER yields moderate improvements in weighted blocking probability.

7.5.4 Experiment 4: Variable Session Sizes

In this last set of simulations, the number of receivers that participate in the arriving session is selected uniformly at random from the set of integers ranging from 25 to 75. Node capacity is fixed at 4 units and session rates are fixed at 1 unit. Examination of the plots in Figure 7.6 reveals that with swapping, CLUSTER and DISPERSE yield approximately the same blocking probabilities, or CLUSTER's blocking probability is marginally lower. When swapping is not permitted, the blocking probability of CLUSTER is significantly higher than DISPERSE. We again conclude that if swapping is enabled, the decision to use CLUSTER or DISPERSE is rather arbitrary. When swapping is not enabled, DISPERSE should be used.



(a) high maximum hop-depth

(b) low maximum hop-depth

Figure 7.6: Heterogeneous setting, variable session size

7.5.5 Reservation Policy

Unlike the telecommunications area in which reservation schemes lead to reductions in blocking of sessions, the reservation scheme analyzed here do not exhibit similar performance gains. This is in part due to the fact that the use of low-bandwidth end-users as forwarding agents required reservations of up to 25% of the user's bandwidth (i.e., one forwarding capacity unit). It will be interesting to evaluate these reservation schemes in environments in which the end users' bandwidth capabilities are larger such that the reservation threshold can be reduced to a smaller fraction on the order of 10%. In that case, we suspect that a reservation scheme can lead to significant decreases in blocking probabilities.

7.6 Simulations with Dynamic Group Membership

In this section we present results of simulations conducted with dynamic node membership. Individual nodes join and leave the sessions, and we use the dynamic versions of CLUSTER and DISPERSE described in the end of Section 7.2. We simulate two scenarios: in one, the number of *sessions* is fixed, but individual nodes join to and leave from this static set of sessions. In the other, the set of active sessions themselves as well as the membership within those sessions vary over time. For all experiments, the maximum session size is 100 nodes, and each session's transmission utilizes one unit of bandwidth.

For the static session case, the total number of sessions is fixed at 10, the join process is Poisson with rate 100 and the inter-update interval equal to 10. For the dynamic session case, the session arrival process is Poisson with rate 1, the session duration is exponentially distributed with a mean of 5 and the join rate within each session is again 100. The inter-update interval for the dynamic session case is much smaller, 0.05, as the topology in this scenario is likely to

evolve much faster than in the static session case. For both scenarios, the member sojourn time is exponentially distributed, and we vary the mean along the x -axis, where the value indicated along the x -axis is the join rate divided by the leave rate. We plot the member blocking probability along the y -axis. We evaluate both the static and dynamic session case using homogeneous node capacities of 4, as well as heterogeneous node capacities uniformly distributed among $\{3, 4, 5\}$.

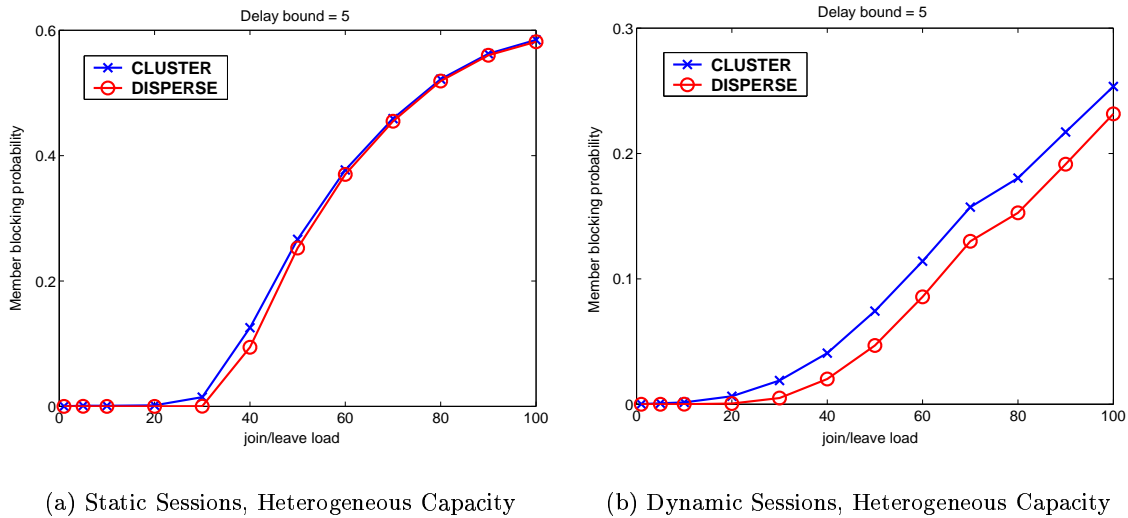


Figure 7.7: Dynamic join/leave experiments, Heterogeneous capacities

In Figure 7.7(a), we see that the blocking rates that result from applying CLUSTER and DISPERSE are similar, especially when the join rate divided by the leave rate is high. Intuitively, this is because under high loads, available bandwidth is scarce, so that the typical tree formed is often the same, whether formed via CLUSTER or DISPERSE. For dynamic sessions, the blocking rates that result from applying DISPERSE are lower than those from applying CLUSTER (Figure 7.7(b)). We observe similar trends in the graphs for higher delay constraints and for homogeneous node capacities. Intuitively, we see two reasons why this is so. First, since in CLUSTER a node is either a leaf node, or (with high probability) completely utilized, a leave operation leads to a bursty process of re-attachment of the subtrees. Either there is no subtree to be re-attached, or there are a high number of nodes that must be re-attached, the latter requiring more bandwidth units. In contrast, DISPERSE leaves “holes” spread out among nodes. Hence, arriving sessions are more easily accommodated. Second, the departure of nodes yields a topology in which DISPERSE is designed to operate. Hence arriving sessions are more naturally accommodated by it as opposed to CLUSTER which needs an update operation before it can best utilize existing resources.

7.7 Conclusion

In this chapter, we have explored the problem of building depth-bounded multicast trees for multiple sessions in networking systems where tree depth and a node's outgoing bandwidth constrain the permitted topology of the tree. We consider two algorithms that build overlay trees within these constraints, one tries to cluster a node's available bandwidth within a single tree, the other tries to disperse the available bandwidth among multiple trees. We derive a lower bound for the blocking probability of algorithms in this networking environment and compare the performance of our algorithms to this lower bound through simulation.

An interesting finding of this study is that the clustering algorithm provably minimizes the session blocking rate in homogeneous network environments. On the other hand, we find through simulation that the dispersing algorithm exhibits a lower blocking rate in heterogeneous networking environments where session sizes, session rates or node capacities differ. Furthermore, the dispersing algorithm performs better in both homogeneous and heterogeneous settings in environments where session participants join and leave in the middle of a session.

In summary, our study indicates that it is more efficient to spread each node's forwarding capacity across sessions given the heterogeneous nature of real networking environments.

Chapter 8

Conclusions and Future Work

With the extremely fast development of the Internet, both in the infrastructure and in the applications, the requirement to enhance the best-effort service provided by current Internet protocols has become a crucial issue, especially for multimedia applications. Many paradigms have been proposed ranging from those that add substantial modifications in the IP layer to those that implement specialized protocols at the application layer.

In this thesis, we have studied two paradigms: Differentiated Services and Overlay Multicast. The former, introduces light changes to the infrastructure and is designed to provide multiple levels of service for any application. The latter, on the other hand, builds protocols upon the existing TCP/IP architecture and is designed to improve the performance of multipoint applications.

8.1 Differentiated Services

We have studied RIO (Random Early Detection with In and Out), a buffer management mechanism that plays a major role in the design and the implementation of the assured service. We have focused on the performance of TCP in interaction with RIO routers. On the one hand, we have developed the expressions of the throughput of a TCP connection marked via token buckets, and on the other, we have used standard queueing analysis to model a network with RIO routers. Through this study, we have found that:

- The linear growth model of the TCP window can serve as a “good” tradeoff between model accuracy and tractability. While the expressions of TCP throughput derived using the linear assumption overestimate the simulated throughput in most scenarios. Simulation results show that the relative error falls into an acceptable range (6%) in most cases, and is less than 30% in all simulated scenarios.
- When the loss probabilities are low, the effect of the bucket size of the token bucket marker on the TCP throughput is negligible. However, this effect is quite significant when the loss probability of *out* packets is high.

- It appears difficult to provide bandwidth assurance to TCP flows that have very heterogeneous characteristics. In particular, TCP connections with small RTTs consume bandwidth beyond their reservation rates, preventing connections with large RTTs from achieving their reservation rates. However, a high drop probability of *out* packets alleviates this problem. Our numerical results show that the fairness among TCP connections increases when the average drop probability of *out* packets increases as well.
- The Tail Drop mechanism is particularly suitable for *in* packets to satisfy various Quality of Service constraints. The drop probability threshold could be then set to a small value to preserve RED objectives such as avoiding global synchronization.
- The drop probability threshold of *out* packets has a significant impact on the TCP throughput and on the average queue length. Setting this parameter consists in trading off between the network utilization and the fairness among TCP connections.

In summary, while we used several simplifying assumptions in our analysis, our results can be used to assist in the design of differentiated services networks.

There are many possible extensions to our work. First, our general loss model (Chapter 3) allows for computing TCP throughput in the case of n levels, though it is unlikely that more than two levels of precedence are useful for the Assured Service. Second, a similar analysis can be carried out for other versions of TCP. Third, in place of assigning a token bucket marker for each TCP connection, it would be more realistic if an aggregation of few *heterogeneous* TCP connections (10 to 50) is marked by a single token bucket.

Finally, another interesting question to investigate is that of short-lived TCP (or HTTP-like) sessions. In particular, we may ask whether such TCP connections could still achieve their reservation rates in the over-booking case, and if so, under what conditions.

8.2 Overlay Multicast

In the first of the two problems we have studied, we have focused on the needs of multicast applications in terms of *delay*. There, we have examined the use of proxies to assist end-systems in building minimized trees. In the second problem, we have focused on *bandwidth* sharing strategies for multiple End-System Multicast sessions. There, we have proposed two algorithms that aim to maximize the number of sessions that can coexist in the same network.

Proxy-Assisted End-System Multicast

Guided by the theoretical framework conducted to examine a hybrid proxy/end-system network, we have developed a heuristic to build depth-bounded overlay trees that minimize the use of proxies. The performance evaluation of the heuristic demonstrates the following:

- In order to minimize the delay, we should give consideration to both transmission delays and bandwidth. One method is to combine both quantities in a weighted sum, and use the weight (α) to control the trade-off between delay and bandwidth according to application requirements. In our simulations we have found that values of α ranging from 30% to 60% optimize proxy usage and tree depth.
- Increasing the number of end-systems raises proxy costs, but at a very slow rate. Further, the increase in the cost approximates a linear growth which means that increasing the number of receivers will not cause an increase in the cost per receiver.
- Increasing the number of proxies decreases costs, but at a rate even slower than the increase in costs when additional end-systems are added to the session. One can conclude that deploying more proxies by the network could be insufficient to reduce costs charged per end-system (client).
- Simulations conducted with the transit-stub network structure shows that when we place proxies at the edges (intermediate nodes) or in the transit portion (core) of the network costs are reduced in comparison to the stubs (nodes connecting end-systems).

Our results can be extended to show that proxies can be temporarily employed during periods of change of group membership. After membership appears more stable, a new tree can be formed with cost re-evaluated. Proxies can also serve as a temporary means of assisting an end-system whose bandwidth suddenly decreases. We assume of course, that a proxy would not exit a session early unless it was no longer needed or some network fault occurred.

There are two basic directions for future work. One involves further improvement of the effectiveness of the heuristic. First, it may be possible to optimize searches for minimum cost within the heuristic by using a binary jump process similar to what is employed within the exact algorithm. Second, we currently fix the weight α during the construction of a tree within the heuristic. It may make sense to vary α during this process, e.g., starting with a very high α and decreasing α during the building of the tree.

The second direction involves investigating proxy placement in networks with varying structure including hierarchical networks (such as the one we used for our simulation) or networks with power law degree distribution. This is especially valuable, considering more recent studies [119] that report that the power law distribution might be more representative of degree nodes of the Internet.

There are also more complex economic issues that can be considered. For instance, we do not consider the case where different service providers charge different rates for their proxy services. Hence, the “best” proxy to minimize the cost need not be the greatest fanout, least delay proxy, but might be the cheapest.

Bandwidth Sharing in End-System Multicast

In order to address the issue of sharing a node's forwarding capacity between multiple overlay sessions, we have developed two algorithms of tree construction. The first algorithm tries to *cluster* a node's available bandwidth among multiple trees (sessions). Intuitively, this algorithm is better at keeping delays small (Lemma 6.2) and at reducing the needs of reconfiguration of sessions. We have proved formally that in a homogeneous setting where all nodes in the overlay network have the same bandwidth and all multicast sessions have the same requirements, the algorithm is optimal. However, the drawback of this algorithm is that it needs swapping of nodes to reduce the blocking probability.

The contrasting algorithm *dispersed* the available bandwidth among multiple trees. If the delay bound is very large this algorithm should perform better. Simulation results reveal that the performance of this algorithm deteriorates drastically if the delay bound is not very large. However, using an iterative dispersing algorithm, we have shown, via simulation, that this strategy performs better in most scenarios.

We have conducted simulations in various heterogeneous settings with static and *dynamic* group membership. Our main findings are:

- Dispersing the bandwidth achieves lower blocking probability when swapping is not permitted. Otherwise, clustering or dispersing the bandwidth leads to similar blocking probability in most cases.
- Reserving 25% of a node's bandwidth for its own transmissions as a source, does not reduce the blocking probability.
- When the number of sessions is constant over time but group membership is dynamic, e.g. a video-conference with simultaneous sessions where people can change the room during the presentations, the clustering and the dispersing algorithm leads to equal performances. In this case, one may prefer to use clustering to simplify management of sessions and access control. For example, when a node breaks down only few sessions will be affected.
- When the number of sessions is dynamic as well as the group membership, e.g. interactive game, then the dispersing algorithm achieves lower blocking probability and should be used. However, the duration between two re-configurations must be carefully chosen to avoid a situation where the tree depth violates the bound for a long time.

There are a number of practical issues that still need to be addressed in order for the proposed algorithms to be implemented within a realistic network setting:

- The overlay connectivity graph may not be fully connected due to communication and state maintenance overheads associated with maintaining connectivity between pairs of nodes.
- Delays and bandwidth constraints can vary as a result of congestion from other sessions competing for the same network bandwidth.

-
- Bandwidth can be constrained at points other than last-mile hops, such that the bandwidth constraints on a pair of overlay edges need not be disjoint.
 - The design of distributed algorithms for dynamically building the trees.

Our on-going work consists in determining how and when it is appropriate to perform the different operations such as node swapping and re-configuration of the overlays. It is also interesting to evaluate different reservation schemes. First, we should study the effect of reserving a small portion of the bandwidth from each node to be used when the delay bound can not be satisfied or when bandwidth is not available. Second, we should find the tradeoff between the reservation cost and the gain in the delay and the blocking probability. In this case, the choice of a good duration between re-configurations is crucial. Finally, we should examine the impact on the delay when reservation can only be handled by few specialized nodes such as proxies.

Bibliography

- [1] Akamai Corporation. Internet Bottlenecks: The Case for Edge Delivery Services, 2000. Akamai whitepaper.
- [2] Mark Allman and Vern Paxson. On estimating end-to-end network path properties. *ACM SIGCOMM'99*, 1999.
- [3] K. C. Almeroth. The Evolution of Multicast: From the MBone to Inter-Domain Multicast to Internet2 Deployment. *IEEE Network Magazine*, Jan 2000.
- [4] Werner Almesberger, Jean-Yves Le Boudec, and Tiziana Ferrari. Scalable resource reservation for the internet. *Internet Draft*, June 1997.
- [5] Werner Almesberger, Jamal Hadi Salim, and Alexy Kuznetsov. Differentiated services on linux. *Internet Draft*, March 1999.
- [6] E. Altman, K. Avrachenkov, C. Barakat, and R. Nunez-Queija. Tcp modeling in the presence of nonlinear window growth. in *Proceedings of Seventeenth International Teletraffic Congress, ITC'17*, Dec, 2001.
- [7] Eitan Altman, Kostia Avrachenkov, and Chadi Barakat. A stochastic model of tcp/ip with stationary random losses. *ACM SIGCOMM'2000*, Aug 2000.
- [8] Francois Baccelli and Dohy Hong. Tcp is max-plus linear. *ACM SIGCOMM'2000*, Aug 2000.
- [9] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable Application Layer Multicast. *SIGCOMM'02*, Aug, 2002.
- [10] C. Barakat, E. Altman, and W. Dabbous. On tcp performance in a heterogeneous network: A survey. *IEEE Communications Magazine*, 38(1):40–46, Jan, 2000.
- [11] F. Bauer and A. Varma. Degree-constrained Multicasting in Point-to-point Networks. In *Proceedings of IEEE INFOCOM'95*, Boston, MA, March 1995.
- [12] Fred Bauer and Anujan Varma. Distributed Degree-Constrained Multicasting in Point-to-Point Networks. Technical Report UCSC-CRL-95-09, UCSC, 1995.

-
- [13] J. C. R. Bennett and H. Zhang. Worst case fair weighted fair queueing. *IEEE INFOCOM'96*, pages 120–128, Mar. 1996.
- [14] Distributed Systems Department Berkeley Laboratory. Remote camera and videoswitcher control software: devserv and camclnt. *Software available at <http://www-itg.lbl.gov/mbone/devserv/>*.
- [15] S. Bhattacharyya, D. Towsley, and J. Kurose. The Loss Path Multiplicity Problem for Multicast Congestion Control. In *Proceedings of IEEE INFOCOM'99*, New York, NY, March 1999.
- [16] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. *Request For Comment: 2475*, December 1998.
- [17] R. Bless and K. Wehrle. IP Multicast in Differentiated Services Networks. *Internet-Draft*, Sep 1999, expired March 2000.
- [18] O. Bonaventure and S. De Cnodder. A rate adaptive shaper for differentiated services. *Request For Comment: 2963*, Oct, 2000.
- [19] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ranakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the internet. *Request For Comment: 2309*, Apr, 1998.
- [20] Tian Bu and Don Towsley. Fixed point approximations for tcp behavior in aqm network. *ACM Sigmetrics 2001, Cambridge, MA USA*, June 2001.
- [21] Ken Calvert and Ellen Zegura. Gt internetwork topology models (gt-itm). *<http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html>*, 1997.
- [22] Jin Cao, William S. Cleveland, Dong Lin, and Don X. Sun. On the nonstationarity of internet traffic. in *Proceedings of Sigmetrics*, pages 102–112, 2001.
- [23] Claudio Casetti and Michela Meo. A new approach to model the stationary behavior of tcp connections. *Proceedings of the 2000 IEEE INFOCOM*, Mar 2000.
- [24] Y. Chait, C. V. Hollot, V. Misra, H. Zhang, and J. C. Lui. Providing throughput differentiation for tcp flows using adaptive two-color marking and two-level aqm. *IEEE INFOCOM'2002*, Jun, 2002.
- [25] Cheng-Shang Chang and Zhen Liu. A bandwidth sharing theory for a large number of http-like connections. *IEEE INFOCOM'2002*, Jun, 2002.

-
- [26] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation Algorithms for Directed Steiner Problems. In *ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, CA, January 1998.
- [27] Y. Chawathe, S. McCanne, and E. Brewer. RMX: Reliable Multicast for Heterogeneous Networks. In *Proceedings of IEEE INFOCOM'00*, Tel-Aviv, Israel, March 2000.
- [28] Yatin Chawathe. Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service. *Ph.D. thesis*, Fall, 2000.
- [29] S. Chen and K. Nahrstedt. An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. *IEEE Network Magazine, Special Issue on Transmission and Distribution of Digital Video*, 12(6):64–79, Nov, 1998.
- [30] Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling Conferencing Applications on the Internet Using an Overlay Multicast Architecture. In *Proceedings of ACM SIGCOMM'01*, San Diego, CA, August 2001.
- [31] Y. Chu, S. Rao, and H. Zhang. A Case for End System Multicast. In *Proceedings of ACM SIGMETRICS'00*, Santa Clara, CA, May 2000.
- [32] D. Clark and J. Wroclawski. An approach to service allocation in the internet. *talk by D. Clark in the Int-Serv WG at the Munich IETF*, August 1997. Also, Internet Draft, Jul 1997.
- [33] David D. Clark and Wenjia Fang. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Transactions on Networking (TON'98)*, 6(4):362–373, Aug, 1998.
- [34] David D. Clark and Wenjia Fang. Explicit allocation of best effort packet delivery service. *Internet Draft*, Septembre 1997.
- [35] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.
- [36] Steve Deering. Multicast Routing in Internetworks and Extended Lans. *Proceedings of the ACM SIGCOMM'88*, pages 55–64, Aug, 1988.
- [37] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. *SIGCOMM'89*, 19(4):1–12, Sep, 1989.
- [38] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Network Magazine*, Jan/Feb 2000.
- [39] Christophe Diot, Walid Dabbous, and Jon Crowcroft. Multipoint Communication: A Survey of Protocols, Functions and Mechanisms. *IEEE Journal on Selected Area in Communication (JSAC)*, 15(3), Apr, 1997.

-
- [40] Constantinos Dovrolis, Dimitrios Stiliadis, and Parameswaran Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. *ACM SIGCOMM'99*, Sep, 1999.
- [41] Kevin Fall and Sally Floyd. Simulation-based comparisons of tahoe, reno and sack tcp. *ACM Computer Communication Review*, Jul, 1996.
- [42] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-Law Relationships of the Internet Topology. In *Proceedings of SIGCOMM'99*, Cambridge, MA, September 1999.
- [43] W. Fang, N. Seddigh, and B. Nandy. A time sliding window three colour marker. *Request For Comment: 2859*, Jun, 2000.
- [44] Wenjia Fang. Differentiated services: Architecture, mechanisms and an evaluation. *PhD thesis, The university of Princeton, Departement of Computer Science*, Nov, 2000.
- [45] W. Feng, D. Kandlur, D. Saha, and K. Shin. Blue: An alternative approach to active queue management. *NOSSDAV'2001*, Jun, 2001.
- [46] W. Feng, D. Kandlur, D. Saha, and K. Shin. Techniques for eliminating packet loss in congested tcp/ip network. *University of Michigan CSE-TR-349-97*, Nov, 1997.
- [47] Paul Ferguson. Simple differential services. *Internet Draft*, November 1997.
- [48] Azeem Feroz, Shivkumar Kalyanaraman, and Amit Kumar. A tcp-friendly traffic marker for ip differentiated services. *Eighth International Workshop on Quality of Service IWQoS'2000*, Jun, 2000.
- [49] Victor Firoiu and Marty Borden. A study of active queue management for congestion control. *INFOCOM 2000*, Mar 2000.
- [50] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker. Adaptive red: An algorithm for increasing the robustness of red's active queue management. *Under submission. Available at <http://www.icir.org/floyd/red.html>*, Aug, 2001.
- [51] Sally Floyd and Van Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–38, Aug, 1995.
- [52] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.
- [53] Paul Francis. Yoid: Extending the Internet Multicast Architecture. <http://www.aciri.org/yoid>, Apr, 2000.
- [54] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, 1979.

-
- [55] L. Gautier, C. Diot, and J. Kurose. End-to-end Transmission Control Mechanisms for Multiparty Interactive Applications in the Internet. In *Proceedings of IEEE INFOCOM'99*, New York, NY, March 1999.
- [56] A. Goel, K.G. Ramakrishnan, D. Kataria, and L. Logothetis. Efficient Computation of Delay-sensitive Routes from One Source to All Destinations. In *Proceedings of IEEE INFOCOM'01*, Anchorage, AK, April 2001.
- [57] E. Hahne. Round robin scheduling for fair flow control. *Ph.D. thesis, Dept. Elect. Eng. And Comput. Sci., M.I.T.*, Dec, 1986.
- [58] M. Handeley. Session directory. university college london. *Software Available at ftp://cs.ucl.ac.uk/mice/sdr.*
- [59] M. Handley and J. Crowcroft. Network text editor (nte): A scalable shared text editor for the mobone. *SIGCOMM'97*, Sep, 1997.
- [60] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured forwarding phb group. *Request For Comment: 2597*, June 1999.
- [61] J. Heinanen and R. Guerin. A single rate three color marker. *Request For Comment: 2697*, Sep, 1999.
- [62] J. Heinanen and R. Guerin. A two rate three color marker. *Request For Comment: 2698*, Sep, 1999.
- [63] Chris Hollot, Vishal Misra, Don Towsley, and Weibo Gong. A control theoretic analysis of red. *INFOCOM 2001*, Apr 2001.
- [64] Inktomi Corporation. The Inktomi Overlay Solution for Streaming Media Broadcasts. Inktomi whitepaper.
- [65] Source: Internet software consortium (<http://www.isc.org/>). (<http://www.isc.org/>), Jul, 2002.
- [66] H. Zhang J. C. R. Bennett. Hierarchical packet fair queueing algorithms. *IEEE/ACM Transactions on networking*, 5(5), Oct, 1997.
- [67] V. Jacobson. Differentiated services architecture. *talk in the Int-Serv WG at the Munich IETF*, August, 1997.
- [68] V. Jacobson and S. McCanne. Visual audio tool. lawrence berkeley laboratory. *Software available at ftp://ftp.ee.lbl.gov/conferencing/vat.*
- [69] V. Jacobson, K. Nichols, and K. Poduri. An expedited forwarding phb. *Request For Comment: 2598*, June 1999.

- [70] V. Jacobson, K. Nichols, and L. Zhang. A two-bit differentiated services architecture for the internet. *Internet Draft, November 1997, then A Request For Comment 2638*, Jul, 1999.
- [71] Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM*, 18(14), Aug, 1988.
- [72] R. Jain, K. Ramakrishnan, and D.-M. Chiu. A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer. *ACM SIGCOMM*, 18(14), Aug, 1988.
- [73] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of USENIX*, San Diego, CA, October 2000.
- [74] Samuel Karlin and Howard M. Taylor. *A First Course in Stochastic Processes, (page 253)*. Academic Press, INC., second edition edition, 1975.
- [75] Frank Kelly. Mathematical modelling of the internet. Available at <http://www.statslab.cam.ac.uk/frank/>, 1999. Also extended version in Mathematics Unlimited - 2001 and Beyond Springer Verlag, 2000.
- [76] Leonard Kleinrock. *Queueing Systems. Volume II: Computer Applications*. John Wiley & Sons, Inc., 1976.
- [77] Vachaspathi P. Kompella, Joseph C. Pasquale, and George C. Polyzos. Multicast Routing for Multimedia Communication. *IEEE/ACM Transactions on Networking*, 1(3):286-292, 1993.
- [78] V. Kumar. *MBone: Interactive Multimedia on the Internet*. New Riders, 1996.
- [79] P. Kuusela, P. Lassila, J. Virtamo, and P. Key. Modeling red idealized tcp sources. *9th IFIP Conference on Performance Modeling and Evaluation of ATM & IP Networks*. Budapest, June 2001.
- [80] P. Kuusela and J. T. Virtamo. Modeling red with two traffic classes. in *Proceedings of Fifteenth Nordic Teletraffic Seminar, Lund, Sweden*. pp. 271-282, August 2000.
- [81] Dong Lin and Robert Morris. Dynamics of random early detection. *ACM SIGCOMM'97*, 27(4), Oct, 97.
- [82] Z. Liu, P. Nain, and D. Towsley. Sample Path Methods in the Control of Queues. *Queueing Systems, Special Issue on Optimal Control in Queueing Systems*, 21, 1995.
- [83] Steven H. Low. A duality model of tcp and queue management algorithms. *ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management*, Sept, 2000.

- [84] F. Lyonnet. Rendez-vous: A new internet videoconferencing tool, inria sophia-antipolis. *Software available at <http://www.lyonnet.org/IVStng/>*.
- [85] J. Mahdavi and S. Floyd. Tcp-friendly unicast rate-based flow control. *Note sent to end2end-interest mailing list*, 1997.
- [86] N. Malouch, Z. Liu, D. Rubenstein, and S. Sahu. A Graph Theoretic Approach to Bounding Delay in Proxy-Assisted, End-System Multicast. In *Proceedings of the International Workshop on Quality of Service (IWQoS)*, Miami Beach, FL, May 2002.
- [87] Naceur Malouch and Zhen Liu. On steady state analysis of tcp in networks with differentiated services. in *Proceedings of Seventeenth International Teletraffic Congress, ITC'17.*, December 2001.
- [88] Naceur Malouch and Zhen Liu. Performance analysis of tcp with rio routers. *IEEE GLOBECOM'2002*, Nov, 2002.
- [89] M. Mathis, J. Semake, J. Mahdavi, and T. Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *Computer Communication Review*, July 1997.
- [90] M. May. *Évaluation Quantitative des Nouveaux Mécanismes de gestion de la Qualité de Service dans Internet*. PhD thesis, Université de Nice Sophia Antipolis, Oct, 1999.
- [91] Martin May, Jean-Chrysostome Bolot, Alain Jean-Marie, and Christophe Diot. Simple performance models of differentiated services schemes for the internet. *IEEE Infocom'99, New York, NY*, March 99.
- [92] Martin May, Thomas Bonald, and Jean Bolot. Analytic evaluation of red performance. *INFOCOM'2000*, Mar 2000.
- [93] S. McCanne. A distributed whiteboard for the network conferencing. *Class Report, UC Berkeley*, May, 1992.
- [94] S. McCanne and V. Jacobson. A flexible framework for packet video. *ACM Multimedia'95*, pages 511–522, Nov, 1995.
- [95] S. McCanne, V. Jacobson, and M. Vetterli. Receiver Driven Layered Multicast. In *Proceedings of SIGCOMM'96*, Stanford, CA, August 1996.
- [96] Vishal Misra, Wei-Bo Gong, and Don Towsley. A fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red. *SIGCOMM'2000*, Aug 2000.
- [97] S. Murphy. Diffserv additions to ns-2. <http://www.teltec.dcu.ie/~murphys/ns-work/diffserv/>, 1999.

-
- [98] B. Nandy, N. Seddigh, P. P. P. P., and J. Ethidge. Intelligent traffic conditioners for assured forwarding based differentiated services networks. *Networking2000*, May 2000.
- [99] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers. *Request For Comment: 2474*, December 1998.
- [100] T.J. Ott, T. V. Lakshman, and L. H Wong. Sred: Stabilized red. *IEEE INFOCOM'99*, pages 1346–1355, Mar, 1999.
- [101] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling tcp throughput: A simple model and its empirical validation. *ACM SIGCOMM'98*, 1998.
- [102] Mehrdad Parsa, Qing Zhu, and J. J. Garcia-Luna-Aceves. An Iterative Algorithm for Delay-constrained Minimum-cost Multicasting. *IEEE/ACM Transactions on Networking*, 6(4):461–474, 1998.
- [103] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An Application Level Multicast Infrastructure. In *3rd Usenix Symposium on Internet Technologies and systems (USITS)*, March 2001.
- [104] J. Postel. Transmission control protocol. *Request For Comment: 793*, Sep, 1981.
- [105] K. Ramakrishnan and S. Floyd. A proposal to add explicit congestion notification (ecn) to ip. *Request For Comment: 2481*, Jan, 1999.
- [106] S. Ramanathan. Multicast Tree Generation in Networks with Asymmetric Links. *IEEE/ACM Transactions on Networking*, 4(4), 1996.
- [107] The robust audio tool (rat). university college london, computer science department. *Software Available at <http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/>*.
- [108] L. Rizzo. pgmcc: A TCP-friendly Single-Rate Multicast Congestion Control Scheme. In *Proceedings of ACM SIGCOMM'00*, Stockholm, Sweden, September 2000.
- [109] Keith W. Ross. *Multiservice Loss Models for Broadband Telecommunication Networks*. Springer-Verlag, 1995.
- [110] Sambit Sahu, Philippe Nain, Don Towsley, Christophe Diot, and Victor Firoiu. On achievable service differentiation with token bucket marking for tcp. *Proc. ACM Sigmetrics 2000, Performance Evaluation Review, Vol. 28, No. 1*, June 2000.
- [111] Sambit Sahu, Don Towsely, and Jim Kurose. A quantitative study of differentiated services for the internet. *IEEE GLOBECOM'99*, pages 1808–1817, Dec 1999. Also invited paper at Performance'99. Istanbul, Turkey.

-
- [112] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. *Request For Comment: 2212*, Sep, 1997.
- [113] S. Shi and J. Turner. Dimensioning Server Access Bandwidth and Multicast Routing in Overlay Networks. In *Proceedings of NOSSDAV'01*, Port Jefferson, NY, June 2001.
- [114] S. Shi and J. Turner. Routing in Overlay Multicast Networks . In *Proceedings of IEEE INFOCOM'02*, New York, NY, June 2002.
- [115] Network simulator ns-2. <http://www.isi.edu/nsnam/ns/>, 1999.
- [116] W. Stevens. Tcp slow start, congestion avoidance, fast retransmit and fast recovery algorithms. *Request For Comment: 2001*, Jan, 1997.
- [117] I. Stoica and H. Zhang. Providing guaranteed services without per flow management. *ACM SIGCOMM'99*, 29:81–94, 1999.
- [118] H. Takahashi and A. Matsuyama. An approximate solution for the steiner problem in graphs. *Math. Japonica*, 24(6):573–577, 1980.
- [119] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topologies, power laws, and hierarchy. *Technical Report 01-746. Computer Science Department. University of Southern California*, 2001.
- [120] D. Tennenhouse, J. Smith, D. Sincoskie, D. Wetherall, and G. Minden. A survey of active network research. *IEEE communications Magazine*, 35(1):80–86, Jan, 1997.
- [121] T. Turletti. Inria videoconferencing system (ivs). *Software available at <http://www-sop.inria.fr/rodeo/personnel/Thierry.Turletti/>*.
- [122] S. Vutukury and J.J. Garcia-Luna-Aceves. A Practical Framework for Minimum-Delay Routing in Computer Networks. *Journal of High Speed Networks*, 8(4), 1999.
- [123] B. Wang and J. Hou. Multicast Routing and its QoS Extension. *IEEE Network*, Jan/Feb 2000.
- [124] Z. Wang and J. Crowcroft. Bandwidth-delay Based Routing Algorithms. In *IEEE Globecom'95*, November 1995.
- [125] Zheng Wang. User-share differentiation (usd) scalable bandwidth allocation for differentiated services. *Internet Draft*, May 1998.
- [126] Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9): 1617-1622, 1988.
- [127] J. Widmer and M. Handley. Extending Equation-Based Congestion Control to Multicast Applications. In *Proceedings of ACM SIGCOMM'01*, San Diego, CA, August 2001.

-
- [128] J. Wroclawski. Specification of the controlled-load network element service. *Request For Comment: 2211*, Sep, 1997.
 - [129] J. Wroclawski. The use of rsvp with ietf integrated services. *Request For Comment: 2210*, Sep, 1997.
 - [130] Ikjun Yeom and A. L. Narasimha Reddy. Modeling tcp behavior in a differentiated services network. *TAMU ECE Technical Report*, May 1999.
 - [131] Ellen Zegura, Ken Calvert, and Samrat Bhattacharjee. How to model an internetwork. *Proceedings of IEEE Infocom'96*, 1996.
 - [132] Lixia Zhang. A new architecture for packet switching network protocols. *Ph.D. thesis, Dept. Elect. Eng. and Comput. Sci., M.I.T.*, Aug, 1989.
 - [133] Q. Zhu, M. Parsa, and J.J. Garcia-Luna-Aceves. A Source-based Algorithm for Delay Constrained Minimum Cost Multicasting. In *Proceedings of IEEE INFOCOM'95*, Boston, MA, March 1995.