

# 5

## Network Dimensioning

---

### 5.1 Simple compressed voice flow model

#### 5.1.1 Model of popular voice coders

Most recent vocoders implement a voice activity detection algorithm, which uses significantly less bandwidth in silence periods than in active speech periods.

For most coders the activity bitrate is constant, and we call it  $M$ . In silence periods very basic coders will have a null bitrate, but coders such as G.723.1 or G.729 actually send some information to describe the background noise level and other parameters. We will call the bitrate during silence periods  $m$ .

$m$  and  $M$  values for popular coders are shown in Table 5.1 for IP packets containing about 30 ms of speech, including all IP overhead.

In Table 5.1, one may wonder why G.723.1 in its 6.4-kbit/s mode has an  $M$  value of 16 kbit/s. This is because 6.4 kbits per second is the bitrate at the output of the coder and does not include transport overheads, such as RTP/UDP/IP headers. We want to properly dimension IP links, and therefore we must take into account such overheads. Table 5.2

**Table 5.1**  $M$  and  $m$  values for common voice coders, when using 30-ms IP packets

Codec	Frames/IP packet	$M$ (kbit/s)	$m$ (kbit/s)
G.723.1 (5.3 kbit/s)	1	16	11.73
G.723.1 (6.4 kbit/s)	1	17.07	11.73
G.729	3	18.67	13.87
Lucent SX7003P	2	20.27	13.87

**Table 5.2** Detailed calculation of the average bitrate, with and without RTP compression

IPv4 overhead (octets)	20						
UDP overhead (octets)	8						
RTP overhead (octets)	12						
Activity rate (%)	35						
		<b>G.723.1</b>			<b>SX7003P</b>		<b>G.729.A</b>
		<i>G (5.3 kbit/s)</i>	<i>(6.4 kbit/s)</i>	<i>(silence)</i>	<i>4.8 kbit/s</i>	<i>(silence)</i>	<i>8 kbit/s</i>
Frame size (data octets)		20	24	4	18	6	10
Frame duration (ms)		30	30	30	15	15	10
Frames per IP packet		1	1	1	2	2	3
Bitrate without overhead (kbit/s)		5.33	6.40	1.07	9.60	3.20	8.00
Octets per IP packet		20	24	4	36	12	30
Overhead IPv4+UDP+RTP (octets)		40	40	40	40	40	40
Bitrate with overhead (kbit/s)		16	17.07	11.73	20.27	13.87	18.67
Average bitrate (with activity rate)		<b>13.23</b>	<b>13.60</b>		<b>16.11</b>		<b>15.55</b>
Overhead cRTP (octets)		2	2	2	2	2	2
Bitrate with overhead (kbit/s)		5.87	6.93	1.60	10.13	3.73	8.53
Average bitrate (with activity rate)		<b>9.68</b>	<b>10.05</b>		<b>12.56</b>		<b>12.00</b>

**Table 5.3** IP bitrate calculation formulas

	Coder activity	(silence)
Frame size (data octets)	$FS_a$	$FS_s$
Frame duration (ms)	$FD_a$	$FD_s$
Frames per IP packet	$n_a$	$n_s$
Bitrate without overhead (kbit/s)	$=FS_a * n_a / FD_a$	$=FS_s * n_s / FD_s$
Octets per IP packet	$=FS_a * n_a = \text{Payload}_a$	$=FS_s * n_s = \text{Payload}_s$
Overhead IPv4+UDP+RTP (octets)	$20 + 8 + 12 = Ov$	$20 + 8 + 12 = Ov$
Bitrate with overhead (kbit/s)	$=(\text{Payload}_a + Ov) * 8 / (n_a * FD_a) = M$	$=(\text{Payload}_s + Ov) * 8 / (n_s * FD_s) = m$
Average bitrate (with activity rate)	$=M * \text{act\_rate} + m * (1 - \text{act\_rate})$	

shows the spreadsheet that was used to calculate the results listed in Table 5.1. It is also interesting to read the bitrate results when the IP/UDP/RTP headers are compressed using cRTP to just 2 octets. The template for calculating such values is straightforward (Table 5.3).

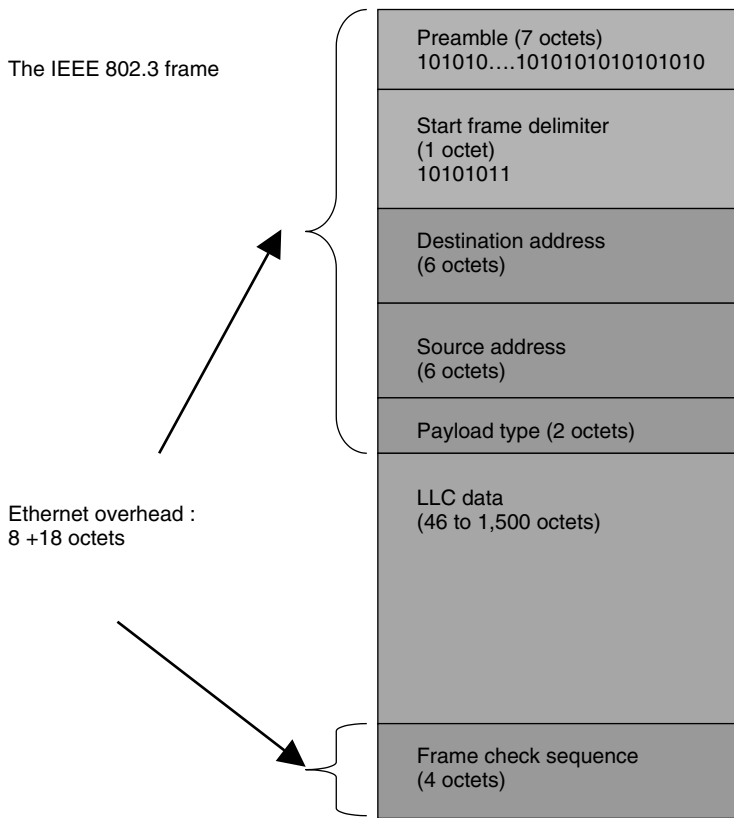
Additional input is needed to calculate the actual bitrate at the physical layer, because IP packets themselves are encapsulated in Ethernet (+26 octets, as shown in Figure 5.1), HDLC (+7 octets), frame relay (+9 octets), PPP frames (+7 octets), or ATM cells using AAL5 (5 header octets for 48 payload octets)!

Figure 5.2 shows network capture of an RTP packet over Ethernet (the Ethernet preamble length and FCS fields do not appear), containing a single frame of G.723.1-encoded voice.

In order to reduce this enormous overhead, multiple compressed voice frames are frequently concatenated in each packet. Of course, packetization delay increases accordingly (number of frames in the packet times the frame delay), so there is a limit to the number of frames that can be accumulated per packet. Mouth-to-ear delay<sup>1</sup> must preserve good interactivity and must be acceptable, given the level of echo cancelation in terminal devices. For low-quality PC-to-phone applications, some vendors accumulate up to 90 ms of speech in each packet (3 G.723.1 frames), but professional use of VoIP should stay in the 20 to 50-ms range.

Given the behavior of most voice coders, we model the one-way network bitrate during a voice conversation by a two-level function characterized by the suite of active speech intervals  $T_{\text{active}}(i)$  and the suite of silence intervals  $T_{\text{idle}}(i)$ , as shown in Figure 5.3. Activity rate  $a$  is defined as the limit when  $i$  gets infinite of  $\sum(T_{\text{active}}) / \sum(T_{\text{idle}} + T_{\text{active}})$ . A good average value is usually 0.35, but in order to be on the safe side we take  $a = 0.5$  in most of our calculations (many VAD detectors remain in active state some time after the actual active period is over, augmenting the  $T_{\text{active}}$  period).

<sup>1</sup> For more details on acceptable mouth-to-ear delays see Chapter 3 on voice quality.



**Figure 5.1** Overhead introduced by Ethernet.

### 5.1.2 Model for $N$ simultaneous conversations using the same coder

When there are  $N$  simultaneous uncorrelated conversations on the same link, they will rarely be active simultaneously and, therefore, the required bandwidth will be less than  $N \cdot M$ .

When  $N$  gets very large it is intuitively obvious that the required bandwidth will be  $N$  times the average bitrate of the coder:  $N \cdot [aM + (1 - a)m]$ . But for small values of  $N$  some calculations are needed.

The probability of  $I$  conversations being simultaneously active among  $N$  can be expressed as:

$$P(I) = C_N^I \cdot a^I \cdot (1 - a)^{N-I} = \frac{N!}{I!(N-I)!} \cdot a^I \cdot (1 - a)^{N-I}$$

(to check that the sum is 1, note the natural development of  $[a + (1 - a)]^N$ ). The results for 30 sessions are shown in Figure 5.4.

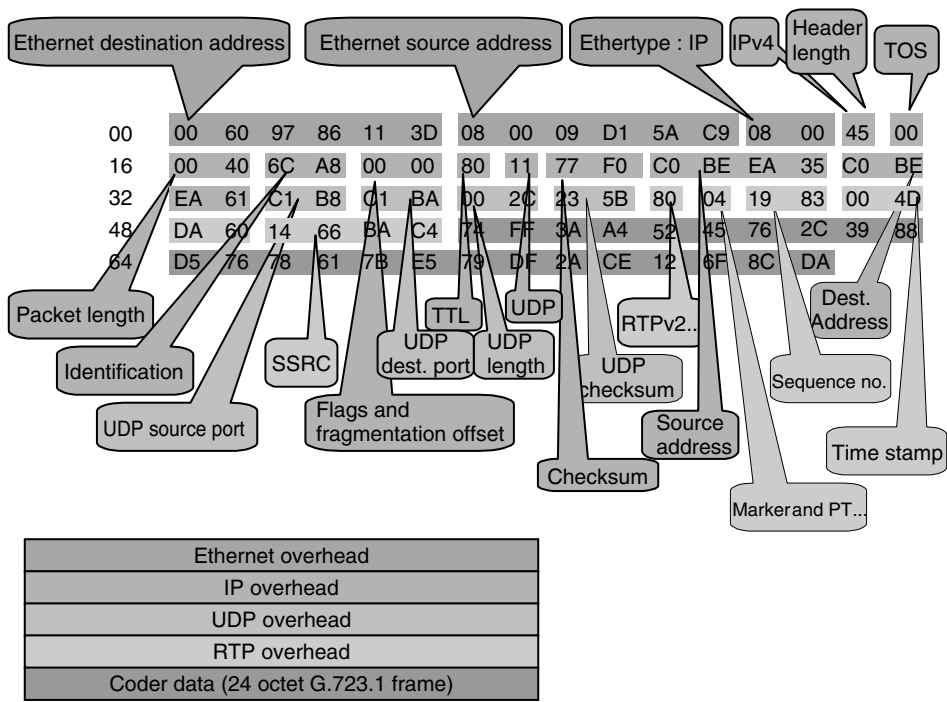


Figure 5.2 Try to find voice in this sea of overheads!.

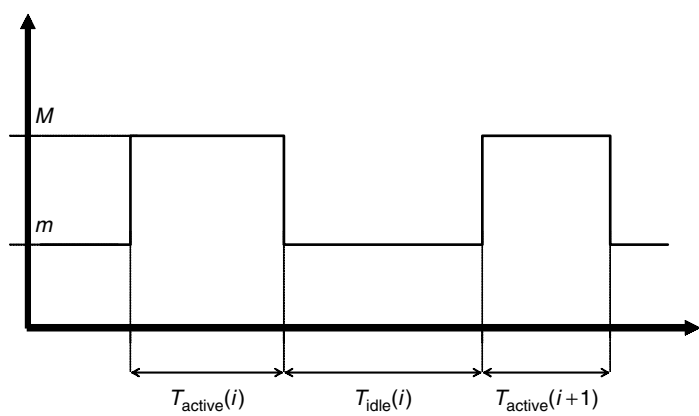
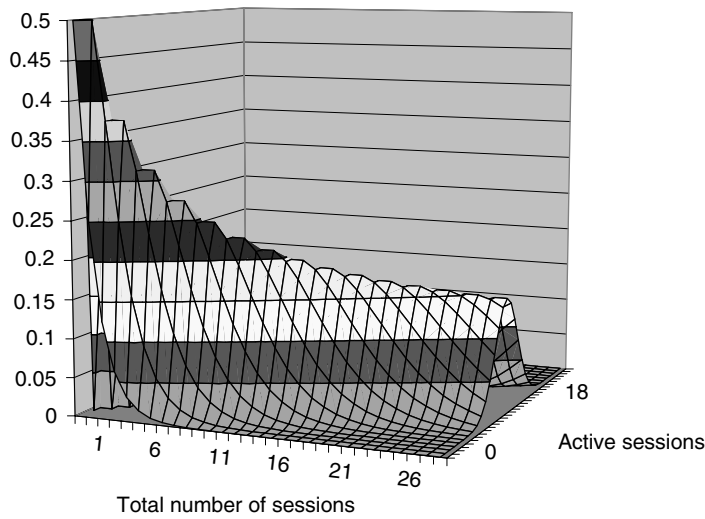


Figure 5.3 Simple on-off voice model.

The average one-way bitrate on a link with  $N$  simultaneous conversations is therefore:

$$Average\_bitrate = \sum_{I=0}^N P(I) * [IM + (N - I)m]$$



**Figure 5.4** Probability of having  $I$  active sessions among  $N$  (for this illustration we took  $a = 0.5$ ).

which can be simplified noticing that:

$$\begin{aligned} \sum_{I=0}^N \frac{N!}{I!(N-I)!} * I x^I * b^{N-I} \\ = x \sum_{I=1}^N \frac{N!}{I!(N-I)!} * I x^{I-1} * b^{N-I} = x \left( \frac{\partial (x+b)^N}{\partial x} \right) = x N (x+b)^{N-1} \end{aligned}$$

and therefore by substituting  $x = a$  and  $b = (1 - a)$ :

$$\sum_{I=0}^N I * P(I) = aN \quad (A)$$

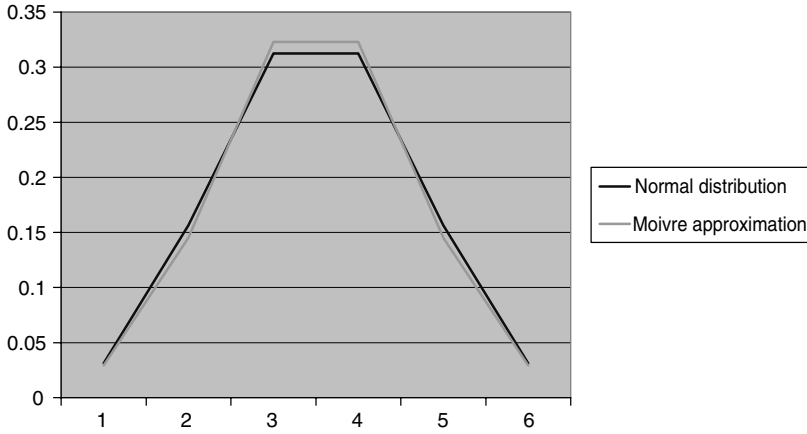
which simplifies our average bitrate expression into:

$$\text{Average\_bitrate} = MaN + Nm - maN = N(Ma + m(1 - a))$$

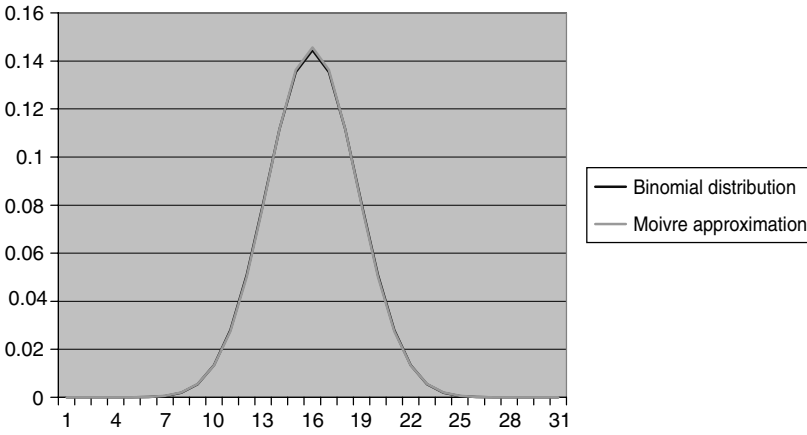
which is the sum of average rates and was intuitively obvious. It can be shown, for such a probability law, that the standard deviation of the number of active session data streams is  $\sigma = \sqrt{Na(1-a)}$ , this value measures the amount of dispersion around the average.

For large values of  $N$ , the probability  $P(I)$  can be approximated as a continuous function, and the Moivre–Laplace theorem shows that:

$$P(I) \rightarrow \frac{1}{\sqrt{2\pi}\sigma} * e^{\frac{-(I-Na)^2}{2\sigma^2}}$$



**Figure 5.5** Exact probability versus Moivre approximation for  $N = 5$ .



**Figure 5.6** Exact probability versus Moivre approximation for  $N = 30$ .

$$P(\text{Bitrate}) = \frac{1}{\sqrt{2\pi}\sigma_b} * e^{\frac{-(\text{Bitrate} - \text{Average\_rate})^2}{2\sigma_b^2}}$$

where  $\sigma_b = (M - m)\sqrt{Na(1 - a)}$ . Compared with the exact calculation, it is much easier to use this function when  $N$  is large (see Figure 5.5 for  $N = 5$  and Figure 5.6 for  $N = 30$ ). This can be found in most worksheets (in Excel this is the NORMDIST function).

### 5.1.3 Loss rate and dimensioning

From the previous results, we will try to evaluate which packet loss rate occurs when a bandwidth  $B$  is reserved on a link and there are  $N$  simultaneous conversations on that

link. Once we know this, it will be possible to calculate the optimal value of  $B$  for a given acceptable loss rate.

To facilitate the reading of the results, we will express  $B$  as a multiple of the elementary average bitrate of a single one-way voice channel during a conversation:  $aM + (1 - a)m$ .

### 5.1.3.1 Loss rate (without queuing)

Each time  $I$  sessions are simultaneously active, the offered bitrate is:

$$Incoming\_bitrate(I) = IM + (N - I)m$$

If we drop packets to reduce the bitrate to  $B$ , the loss rate when  $I$  sessions are simultaneously active is (with fluid approximation):

$$Loss\_rate(I, B) = \max(Incoming\_bitrate - B, 0)$$

and the number of bits lost during a time interval  $T_i$  is:

$$Loss(I, B, T_i) = T_i * Loss\_rate(I, B)$$

Over a time interval  $T$ , we group the packets lost for each subinterval during which  $I$  conversations are simultaneously active (the cumulated duration of each group is  $T_i = P(I)$  if  $T = 1$ ). We can calculate the average packet loss rate by dividing the total traffic lost by the offered traffic during  $T$ :

$$\begin{aligned} Average\_loss\_rate(N, B) &= \left[ \sum_{i=0}^N Loss(I, B, T_i) \right] / \left[ \sum_{i=0}^N T_i (Nm + I(M - m)) \right] \\ &= \frac{\sum_{i=0}^N T_i * \max(Nm + I(M - m) - B, 0)}{[Ma + (1 - a)m] * N * T} \end{aligned}$$

where we use relations (A) to simplify the denominator.

In order to find the minimal bandwidth of a link that has to carry  $N$  calls, one must increase  $B$  until the calculated percentage loss equals the maximum tolerable average loss rate. As an example, Table 5.4 is what we get for the G.729 coder with 3 frames per packet (including IP/UDP/RTP headers, with  $M = 18.67$  and  $m = 13.87$ ) and an activity rate of 0.5, where we considered that average packet loss rate had to be kept below 0.5% from 1 to 15 streams and below 1% for more than 15 streams. On average the bit loss rate and the packet loss rate are equal.

In fact, these results (based on average loss rates) should be considered with care when there are only very few conversations, because the active period at maximum bitrate (maximum loss rate) of the resulting stream can be very long (10 s or more for a single stream), and during this time users will experience voice quality with a loss rate equal to the loss rate when all streams are active, which can be much larger than the average loss rate; this is mitigated in reality by the small buffers of routers and results in jitter,





not packet loss, if not too many data are accumulated in the router buffer. We calculate in Table 5.5 an indication of the data that would accumulate during the peak period (considering that peak period duration is inversely proportional to the number of streams) if the router has sufficient buffers.

Note that not too many data can be accumulated in a buffer, otherwise the corresponding delay creates unacceptable jitter and packets are lost when they arrive at the destination. We need to dimension the link size and the size of router buffers in order to keep the packet loss low enough while not creating excessive jitter. It is acceptable to consider that some packets will be queued in small router buffers (e.g., 2 RTP packets or 60 ms of speech in our sample case of G.729 with 3 frames per packet), creating jitter, and that the buffer will overflow during peak periods, resulting in packet loss. This consideration leads us to suggest a target of 0.5% average loss for 1 to 15 conversations, not 1%, in order to take into consideration user perception of packet loss during peaks.

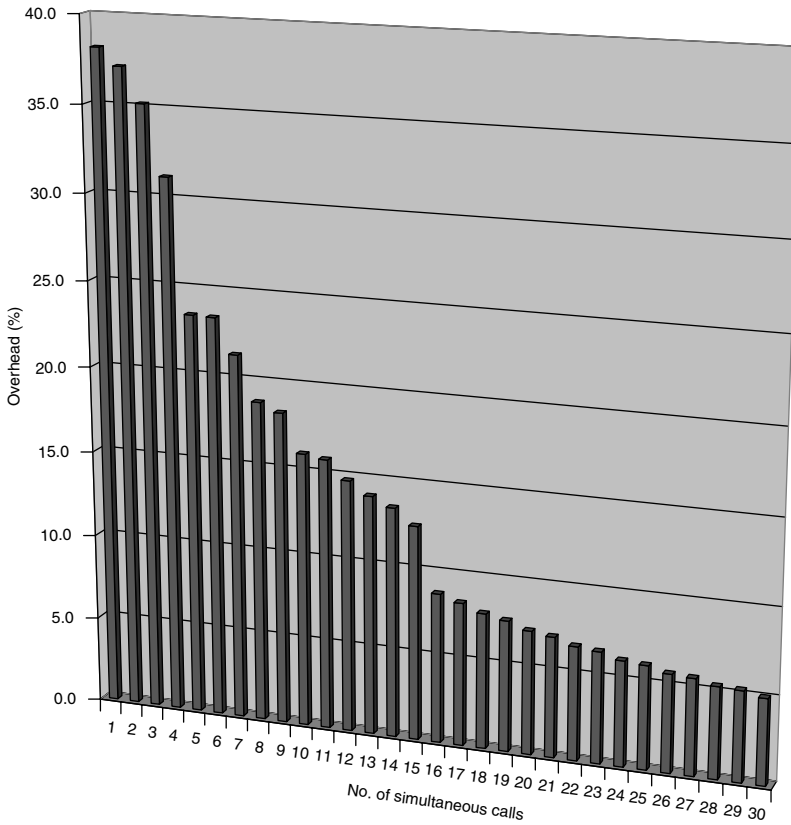
### 5.1.3.1.1 Overhead bandwidth, loss rate distribution

Here we use the term ‘overhead’ for the bandwidth that must be provided in excess of the average rate to accommodate peak traffic. This is unrelated to the IP overhead that was discussed in other sections.

When a link is used to carry only a few conversations, we note that it must be dimensioned almost as if all conversations were always active, which leads to a significant portion of the link bandwidth being unused most of the time. In the example above the maximum overhead (for a single conversation) is only 13.8%, because the  $M$  and  $m$  bitrates are not very different, due to the impact of IP headers on bandwidth. But on a link using compressed RTP ( $M = 8.53, m = 3.73$ ), such as that illustrated in Figure 5.7, the reduction of bandwidth overhead with the number of conversations is much clearer, from 40% to less than 5%. As the number of conversations increases, the link size gets closer to the number of channels times the average bitrate of one channel and the link utilization rate increases. This is because the variance of a sum of  $n$  data streams (which approximates dispersion around the average), varies as  $\sqrt{n}$  and therefore the bandwidth overhead required for an acceptable loss rate, compared with the average total voice bitrate, varies as  $\sqrt{n}/n = 1/\sqrt{n}$ .

**Table 5.5** Assessment of data accumulated in the router buffers during the peak period

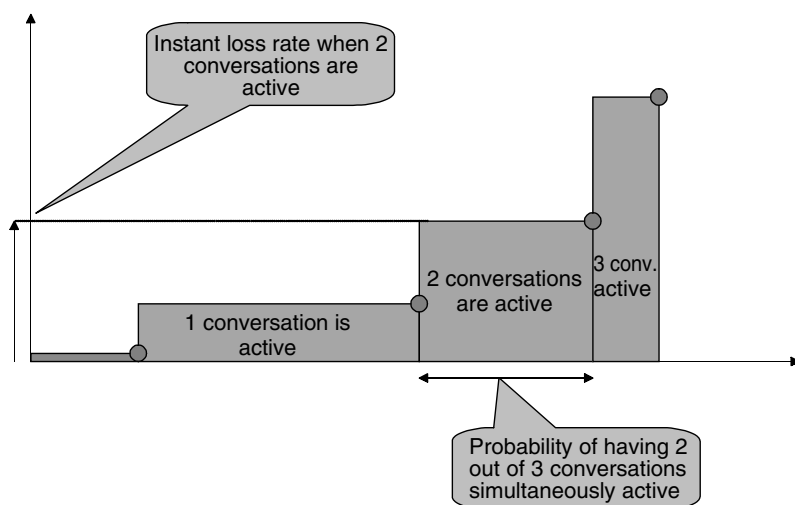
Number of conversations	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Average bitrate	16.3	32.5	48.8	65.1	81.4	97.6	113.9	130.2	146.4	162.7	179.0	195.2	211.5	227.8	244.1
Minimal link bitrate	18.5	36.7	54.1	69.8	87.2	103.4	120.1	136.6	152.9	169.4	185.6	202.2	218.3	234.9	250.9
Equivalent channels	1.1	2.3	3.3	4.3	5.4	6.4	7.4	8.4	9.4	10.4	11.4	12.4	13.4	14.4	15.4
Overhead bitrate	2.2	4.1	5.2	4.7	5.8	5.8	6.2	6.4	6.4	6.7	6.6	6.9	6.7	7.1	6.8
Overhead (%)	13.8	12.8	10.8	7.3	7.2	6.0	5.4	4.9	4.4	4.1	3.7	3.6	3.2	3.1	2.8
Loss (%)	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Cumulative peak loss (kbit)	1.6	3.3	6.5	12.2	12.3	14.3	15.2	16.0	16.9	17.3	18.0	18.2	18.8	18.9	19.4
Duration of peak (s)	10	5	3.333	2.5	2	1.667	1.43	1.25	1.11	1	0.91	0.83	0.769	0.714	0.6667
Peak delay in buffer (ms)	88	89	120	175	142	138	126	117	110	102	97	90	86	81	77



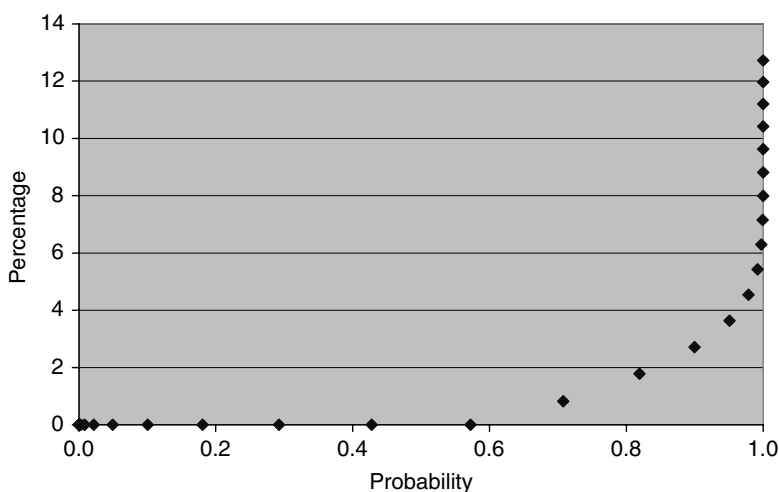
**Figure 5.7** Bandwidth overhead required according to number of simultaneous calls, using cRTP.

As discussed above, we should keep in mind that what we call the ‘loss rate’ is an average value: during a single conversation it will be below this level most of the time, but can be significantly above it for short periods of time. This is illustrated in Figure 5.8: for each rectangle  $i$ , the  $x$ -axis represents the probability that  $i$  conversations are active (have a bitrate of  $M$ ) simultaneously among  $N$  and the  $y$ -axis represents the loss rate when this occurs. The link itself is dimensioned to carry  $N$  conversations with an average loss of 1%.

Figure 5.9 shows the calculated values for  $T = 30$ , using a G.729 coder with 3 frames per packet. We represent only the upper right corner of each rectangle for clarity. This chart can be interpreted as follows: for a conversation of 1,000 s, there will be a 0% loss rate during most of the conversation (810 seconds), a loss rate between 0% and 5% for a cumulated total of 130 s, a loss rate of 5–10% for a cumulated total of 60 s, and more than 10% for no more than a few seconds. Of course, good, bad, and average quality periods will be interleaved in the conversation, and the periods with high loss rates (corresponding to a high bitrate at this instant) become shorter as the loss rate increases.



**Figure 5.8** Probability of higher loss rates is lower. Loss profile on a link dimensioned to carry 3 conversations with an average loss of 1%, when all 3 conversations are established.



**Figure 5.9** Diagram of loss rate probability for 30 active calls (G. 729, 30 ms).

This is important because most IP phones and media gateways recover well from the loss of an isolated packet; therefore, the periods of time corresponding to a very high packet loss rate will go unnoticed if they are very brief. Typically, if the average duration of a period of speech is 10 s, the duration of each occurrence with a bandwidth corresponding to  $N$  active streams simultaneously will be of the order of  $10/N$  s. During this period, and assuming that loss is evenly distributed across all streams the order of magnitude of

the number of packets lost per stream is:

$$Lost\_packets\_per\_stream \approx \frac{10 * Loss\_rate}{N * Packet\_duration}$$

For instance, in the example above, for peaks of 30 simultaneous streams, the occurrence lasts about 0.33 s, corresponding to about 11 packets of 30 ms for each stream. The instant loss rate is about 13%, so about 1.4 packets per stream will be lost. The user is likely to hear a brief glitch, but the periodicity of such a glitch is very low.

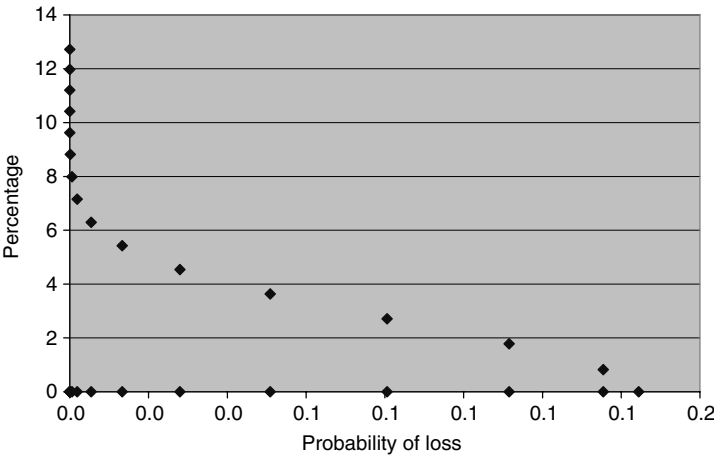
Another way to represent the same information is to plot instantaneous loss rates as a function of their respective probabilities. Figure 5.10 shows that higher loss rates are very unlikely, which is to be expected since the average loss rate is only 1%.

### 5.1.3.1.2 Conclusion

The calculation we have just done is only relevant for packet voice systems with silence compression and assumes there is no buffer in the network (i.e., any excess traffic is immediately discarded). With traditional telephony systems, the only advantage of multiplexing several lines on a link is that not everybody *phones* at the same time, and hence it is possible to provide less capacity than the number of multiplexed lines (this is explained in Section 5.5 on modeling call seizures). This also applies to IP telephony; however, IP telephony (and packet voice technologies in general) can also take advantage of the fact that not everybody *talks* at the same time!

### 5.1.3.2 Loss rate (with queuing)

The previous model only approximates what happens in an IP backbone, because it considers that routers do not buffer packets (packets that cannot be forwarded immediately because the link is congested are immediately discarded). In fact, this is not true, because



**Figure 5.10** Another representation of loss rate probability.

routers do have buffers and therefore the real packet loss rates through an IP backbone would be lower than calculated above. If the provisioned bandwidth is higher than the average bandwidth, buffers convert packet loss into delay, as shown in Figure 5.11.

When, eventually, routers have very large buffers, packets will never be dropped by the network. So in theory we should use a much more complex model that takes into account buffering. This model would also need to take into account the jitter buffer discard policy of receiving terminals. For instance, the following formula holds for a single conversation, where  $b$  is the average duration of active periods,  $C$  the speed at which the buffer is emptied, and  $\varepsilon$  the loss rate:

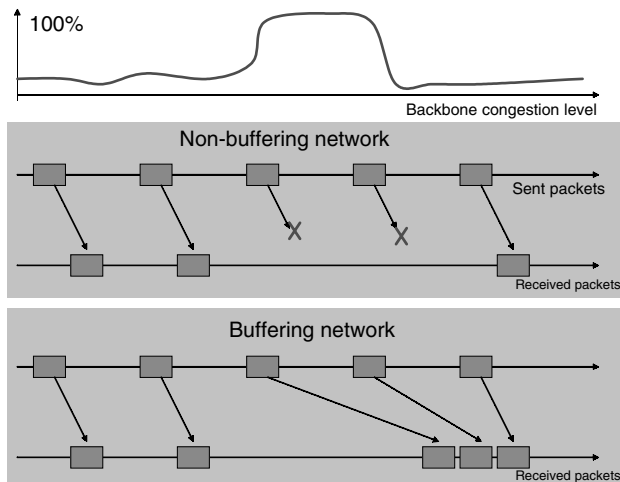
$$0 = \beta \times \exp\left(-\frac{\text{Buffer\_size} \times (C - m - a(M - m))}{b \times (1 - a)(M - C)(C - m)}\right) - \varepsilon = f(\text{Buffer\_size}, C)$$

$$\beta = \frac{(C - m - a(M - m)) + \varepsilon a(M - C)}{(1 - a)(C - m)}$$

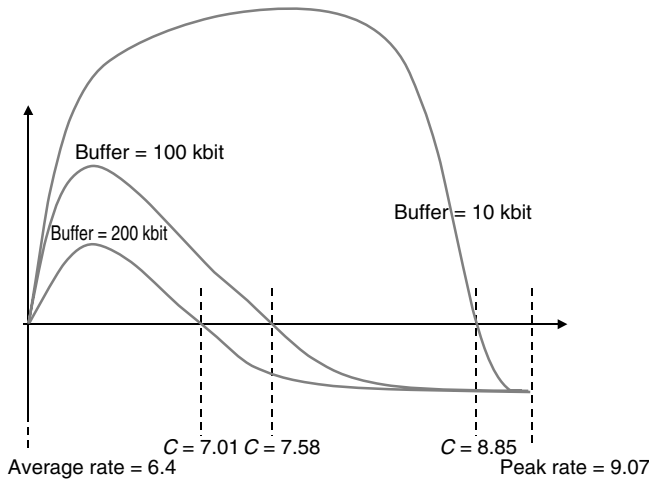
This must be solved numerically to find the appropriate values of  $C$  for a target loss rate. Additional details and an approximation of  $C$  can be found in ‘Equivalent capacity and its application to bandwidth allocation’, by Guerin et al. (1991).

Figure 5.12 ( $M = 9.07$  kbit/s,  $m = 3.73$  kbit/s,  $b = 10$  s,  $\varepsilon = 0.01$ ,  $a = 0.5$ ) shows that  $C$  can be reduced significantly as the size of the buffer increases. But, in our application, real-time telephony, we cannot take advantage of large buffers. The problem is that queued packets get delayed, reducing the interactivity of the conversation, and the increased jitter will cause some packets to be discarded by the jitter buffer of the receiving terminal if the extra delay exceeds a given threshold.

Do we really want to gain a few percent of bandwidth at the expense of reduced interactivity? As we saw in Chapter 3 on voice quality, the greatest constraint of IP telephony is achievable end-to-end delay, which is in most cases at the extreme limit of



**Figure 5.11** Buffers turn packet loss into additional delay.



**Figure 5.12** Required capacity  $C$  for a target loss rate of 1%, according to buffer size.

what is considered acceptable for an average listener. By letting voice packets accumulate in the network, we would increase jitter and force terminals to adapt by building larger jitter buffers: this would lead, inevitably, to additional delays.

Therefore, we think that it is good practice to dimension voice over IP links on the basis of there being no buffering. This leads, for small links, to some overprovisioning, but we will see in the next sections that this overhead can be used by non-real-time traffic.

#### 5.1.4 Packet or frame loss?

Up to now we have calculated the characteristics of the offered bitrate and deduced how much of it could not get through a line of given bandwidth. In doing so we used the ‘fluid approximation’; in fact, routers will not throw away one bit here and there, but whole IP packets. However, this does not change our result which is also valid for the packet loss rate, the bit in excess causes loss of a complete packet (error multiplication), but as a complete packet is pumped from the buffer, other areas that would have experienced bit excess now have available capacity.

Is this the same as the frame loss rate?

If, as we have assumed in our calculations, the frames generated by the coder are simply grouped in IP packets, then the answer is yes. But this is not always true, as some manufacturers use redundancy schemes based on interleaving redundant frames in multiple packets, in order to recover from packet loss. This is often the case for non-standard gateways optimized to work on the Internet, where the bandwidth is ‘free’. The redundancy scheme used by such gateways leads to greater bandwidth usage, but, since it remains only a tiny fraction of overall Internet traffic, it will in most cases not significantly increase the packet loss measured between two gateways.

A basic redundancy scheme would simply be to copy one coder frame in every two IP packets ... if only one IP packet out of two is lost, then the frame can be recovered

and there will be no frame loss. We have seen demonstrations by a manufacturer proudly showing a 50% packet loss with perfect voice quality and doing so using this trick! Of course, a real network is not as simple. The manufacturer's IP network simulator was dropping *exactly* one packet out of 2. The Internet is not so well behaved and will frequently drop entire sequences of packets in a row, so the basic redundancy scheme would not work that well under real conditions!

Our personal advice is to be *very* careful when considering marketing material that proudly announces 'magic' redundancy schemes that beat the competition out of sight. Such material fails to point out that it is most unfriendly to jam the Internet with such a careless use of bandwidth and that there is frequently a delay trade-off when using redundancy.

If, however, you do want to use redundancy, find out how the redundancy scheme performs in the presence of grouped packet loss (or even better, test it). Then look at the packet loss characteristics of the network you will be using (isolated packets or many packets dropped in a row) and decide if the particular redundancy scheme in use will work as advertised. Finally, measure the end-to-end delay that is obtained and check that you are still within the 'acceptable' range.

### 5.1.5 Multiple coders

So far we have only considered one type of coder, with the same value of  $M$  and  $m$  for all voice flows. Different types of customers may require different qualities of service, and therefore different coders may be used.

A possible approach to network dimensioning in this case is to rely on the Gaussian approximation to traffic generated by one class of coders. Remember that, in the case of  $N$  independent and identically distributed flows, traffic distribution could be considered for large values of  $N$  to be Gaussian distribution:

$$P(\text{Bitrate}) = \frac{1}{\sqrt{2\pi}\sigma_b} * e^{\frac{-(\text{Bitrate} - \text{Average\_rate})^2}{2\sigma_b^2}}$$

where  $\sigma_b = (M - m)\sqrt{Na(1 - a)}$ .

An interesting property of the sum of independent variables is that the variance of the sum equals the sum of the variances of each variable ( $\sigma^2 = \sum \sigma_i^2$ ). In addition, if each variable follows a Gaussian law, then the sum is also a Gaussian law.

So an approximation to the traffic distribution in the case of multiple coders is a Gaussian law, with an average value equal to the sum of average values and a variance equal to the sum of variances obtained for each group of conversations with the same type of coder.

At this stage it is easy to calculate what proportion of the traffic is discarded for a given capacity  $C$ :

$$\text{Loss\_bitrate} = \int_C^\infty B * P(B) dB$$



and the loss rate:

$$\begin{aligned}
 \text{Loss\_rate} &= \frac{1}{\sqrt{2\pi}\sigma * \text{Average\_rate}} * \int_C^\infty B * e^{-\frac{(B - \text{Average\_rate})^2}{2\sigma^2}} dB \\
 &= \frac{\sigma}{\sqrt{2\pi} * \text{Average\_rate}} * e^{-\frac{(C - \text{Average\_rate})^2}{2\sigma^2}} + \frac{1}{\sqrt{\pi}} \int_{\frac{C - \text{Average\_rate}}{\sqrt{2}\sigma}}^\infty e^{-x^2} dx \\
 &< \left( \frac{\sigma}{\sqrt{2\pi} * \text{Average\_rate}} + \frac{\sigma}{\sqrt{2\pi}(C - \text{Average\_rate})} \right) * e^{-\frac{(C - \text{Average\_rate})^2}{2\sigma^2}} \\
 &= \left( \frac{\sigma(C - \text{Average\_rate}) + \sigma * \text{Average\_rate}}{\sqrt{2\pi} * \text{Average\_rate} * (C - \text{Average\_rate})} \right) * e^{-\frac{(C - \text{Average\_rate})^2}{2\sigma^2}}
 \end{aligned}$$

For this calculation we used the identity:

$$\int_a^\infty e^{-x^2} dx < \int_a^\infty \frac{x}{a} e^{-x^2} dx = \frac{1}{2a} e^{-a^2}$$

Finding  $C$  for a target loss rate  $\varepsilon$  amounts to calculating the inverse of this function; this can be done numerically or, when the average rate is large compared with the standard deviation (in the first term approximate  $(C - \text{Average\_rate}) \simeq \sigma$ ), using the following approximation (also given by Guerin et al., 1991):

$$C = \text{Mean\_aggregate\_bitrate} + \alpha \times \text{Standard\_deviation\_of\_aggregate\_flow}$$

where  $\alpha = \sqrt{-2 \ln(\varepsilon) - \ln(2\pi)}$ .

## 5.2 Building a network dedicated to IP telephony

### 5.2.1 Is it necessary?

We will see in the following sections that it is feasible, and in fact desirable, to mix all types of IP traffic on one common backbone. But, in order to keep a reasonable quality-of-service level for voice flows, it is necessary to have routers with sophisticated queuing capabilities.

### 5.2.2 Network dimensioning

Suppose we have six sites in three countries that need to communicate using IP telephony. How do we dimension such a network (Figure 5.13)?



**Figure 5.13** Sample VoIP network.

### 5.2.2.1 Traffic matrix

The first step is the same as that in switched telephone networks: we need to know who phones where, how often, and for how long. This information is usually derived easily from existing phone bills during a reference period. We then need to choose an optimal route on the network for each of those calls. For this we need to know the cost of each link per unit of bandwidth (e.g., in the case of a leased line it is monthly fee divided by bandwidth).

The cost of carrying IP traffic from A to B or B to A is not always the same, since a provider could decide to use a satellite link one way. But satellite links add to end to end delay. Let's consider symmetric cost in the context of Table 5.6. This cost matrix could also depend on time, as some providers have discount rates during off-peak hours.

**Table 5.6** Cost matrix of the sample network

From/To	1	2	3	4	5	6	A	B	C
1							2		
2							1		
3									1
4									1
5								1	
6								2	
A	2	1						10	10
B					1	2	10		
C			1	1			10	5	

If there is only ‘on-net’ traffic, then the mapping of each phone call to a route in the network is straightforward: we simply take the least costly route at the time of the call (e.g., a phone call from 1 to 6 will be routed through A and B as opposed to A–C–B). For a more complex cost matrix, the cost-optimized path for reaching each destination from a given origin can be calculated recursively by calculating, for each destination, the number of hops needed to reach it and for which price, then repeating the algorithm for each previous hop and summing the total cost. This results in a number of paths that reach each origin, allowing us to select just the lowest cost one.

Once each call is mapped to a route, we need to calculate the maximum number of simultaneous calls over each link (the so-called ‘busy hour’ traffic).

When there is also ‘off-net’ traffic, the optimal route through the network is calculated by minimizing the ‘on-net’ cost added to the cost of the hop from the last ‘on-net’ node and the final destination over the switched telephone network. In most cases the route ends at the ‘on-net’ node closest to the final destination. But, things can get less rational when the last hop crosses national borders! Some calls (e.g., local calls) may not be routed through the IP network at all.

### 5.2.2.2 Link sizing

Once each call has been mapped to an optimal route, we are able to calculate the number of simultaneous calls on each link at any given time. In general the link bitrate cannot be adjusted dynamically, so we must use the peak number of simultaneous calls (‘busy hour’) as our input to dimension the link and use the Erlang formula (see Section 5.5.4.2 and 5.5.4.3) to plan for variations during this busy hour. At this stage we know the maximum number of simultaneous calls that will be routed on each link.

The difficulty is that the link size also depends on the acceptable average packet loss rate. If we allow for more loss, then the link can be adjusted to a smaller capacity. Table 5.7 shows the bandwidth overhead compared with the average bitrate for a given packet loss (G.729, 3 frames per packet).

For a given coder in a given configuration, it is relatively easy to find the acceptable average packet loss using simple tests. This packet loss can be considered as our end-to-end budget that needs to be split among all the possible paths for a phone call through our network. There are many ways to split this budget, and they are not equivalent. For instance, in Table 5.8 we consider two links: one is a transatlantic line ( $L_2$ ) with a cost

**Table 5.7** Required capacity for a target loss rate

Simultaneous calls	Loss rate (%)					
	3.00	1.00	0.50	0.10	0.01	0
1	9	13	14	15	15	15
5	0	5	7	12	15	15
10	–2	2	4	8	11	15
30	–3	0	1	4	6	15
100	–3	–1	0	2	3	42

**Table 5.8** Costly links should consume most of the packet loss budget

$L_1 =$	0.9%	0.7%	0.3%	0.1%
$L_2 =$	0.1%	0.3%	0.7%	0.9%
Channels on link 1	10			
Channels on link 2	100			
Overhead (link 1)	2.4%	3.0%	5.3%	7.7%
Overhead (link 2)	1.6%	0.7%	-0.2%	-0.5%
Distance on link 1	1			
Distance on link 2	100			
Cost on link 1	10			
Cost on link 2	10,000			
Total overhead cost	163.13	71.10	-20.76	-53.73

of 10,000 for each supplementary unit of bandwidth; the other is a local link ( $L_1$ ) with a cost of 10 for each supplementary unit of bandwidth. This gives us a loss budget of  $L_1 + L_2 = 1\%$ .<sup>2</sup> In this example we see that the best way to split the loss rate is to give most of the loss budget to the transatlantic line.

This issue is very complex in general. The same transatlantic link could serve hundreds of smaller local access links; in this case the cumulative overhead cost of all the local lines could become larger than the cost of the transatlantic link ... This is a nonlinear optimization problem and its resolution is outside the scope of this book. Fortunately, the constant drop of bandwidth costs over long-distance links makes this optimization a bit futile in most cases.

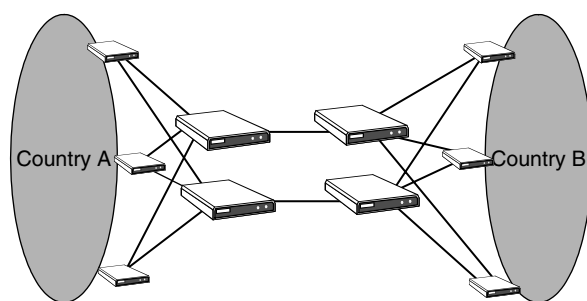
### 5.2.2.3 Fault tolerance

The phone network is a mission-critical system, and the failure of one IP link should not jeopardize corporate communications. This is why a careful network planner will plan ahead for such failures and assess their consequences. This is done by considering the network without this link and rerouting all phone calls according to the new topology (this is done automatically by IP routers). The network must be dimensioned to handle this configuration.

Planning for link outages therefore consists in simulating all possible topologies such that any of their constituent links can fail ( $F(\text{link}_i)$ ) and then dimensioning the remaining links to handle the rerouted flows. The final capacity of a link should be sufficient to compensate for all possible  $F(\text{link}_i)$ .

Some critical backbones may even be dimensioned to handle the simultaneous failure of two links. Note that this step is not necessary if the layer 2 protocol already has such fault protection (e.g., SDH rings, etc.) and is already dimensioned to support rerouted traffic.

<sup>2</sup> Loss rates are not additive. For example, in Table 5.8  $(1 - L_2)(1 - L_1) = 0.99$ ; however, for small values of  $L_1$  and  $L_2$  the  $L_1 * L_2$  term is negligible.



**Figure 5.14** Fully fault-tolerant IP network.

A potentially more significant problem is the loss of a router. If the router connects  $N$  links, then the situation is the same as if all these links were down. As this is a layer 3 failure, it will not be recovered by layer 2 security mechanisms (SDH rings, etc.). This type of failure can generally be avoided if network providers use redundant configurations such as the one shown in Figure 5.14, where traffic can always be rerouted even if one router goes down.

## 5.3 Merging data communications and voice communications on one common IP backbone

The previous sections show that it is necessary to plan for some overhead capacity when designing an IP network dedicated to IP telephony. It is very tempting to use this spare capacity for best effort data when it is unused by voice flows. But, what are the consequences?

### 5.3.1 Prioritization of voice flows

The first condition, in order to carry voice flows over a general purpose IP backbone, is to guarantee that packet loss and network delays and jitter will be kept to a minimum for voice packets. There are several ways of achieving this:

- Do nothing: TCP traffic backs off when facing network congestion. Therefore, UDP traffic (and RTP over UDP) will tend to occupy whatever bandwidth it needs at the expense of TCP traffic. However, this adaptation of TCP traffic is rather slow and works by ‘trial and error’: it sends traffic first and interprets packet loss as congestion. Therefore, TCP traffic will always grow until it congests the network, and having done so backs off again. It will maintain the network in near-congestion state and cause some marginal packet loss on UDP traffic in order to ‘get a feel’ of the state of network congestion. Moreover, there are some TCP stacks that do not respect this back-off strategy. All considered, relying on TCP congestion control algorithms in order to prioritize UDP is not safe to build reliable VoIP networks, even on private networks.

- **Prioritize all UDP traffic:** This is the easiest way to prioritize voice flows, since IP telephony RTP packets are carried over UDP. The side-effect is that all other UDP flows (such as DNS queries) are also prioritized. This is fine in general, because most applications written on top of UDP need minimal delays. In public networks, however, this is a very dangerous practice. Soon, some bright spark will discover that it is actually quite simple to simulate a TCP connection over UDP, and chances are that after a while more and more customers will begin to use all sorts of tricks to send most of their traffic (e.g., peer to peer), not just voice, over UDP! Our advice is to use this method only on corporate backbones, where backbone traffic is well understood and connected networks are well behaved.
- **Use IP precedence levels (DiffServ):** Many routers can be configured to use IP precedence, or DS, information (see Chapter 4 on QoS for more details) in IP packets to prioritize classes of traffic. Routers can be configured in several different ways:
  - Assigning a minimal bandwidth to each class.
  - Assigning a weight to each class and sharing the available bandwidth between classes that have a backlog proportional to these weights.
  - Giving head-of-line priority to a class.

Each method is discussed in detail in Chapter 4. The way in which many of them are used depends on the other flows that are being carried on the network. It is important, when evaluating one method or the other, to verify the behavior of the scheduling algorithm being used regarding delays. We have found it sometimes very difficult to obtain such data from manufacturers. It is not enough just to have the name of the algorithm (e.g., WFQ is now used as a marketing term by many manufacturers for all scheduling algorithms that exhibit selective prioritization properties).

It is also necessary not to ignore the behavior of level 2 multiplexing algorithms: if your IP network is built on top of plain frame relay links (without priority extensions), you may prioritize some IP packets at the IP level... but the frame-relay switches will gladly ignore this information! Connectivity providers using ATM, MPLS, or SDH backbones are likely to offer better support for differentiated classes of service, but it is always useful to check delays and jitter over a shared backbone before trying to use it for IP voice.

### **5.3.1.1 Example configuration**

Let's build a backbone supporting three classes of service:

- Real-time class for voice.
- Committed bandwidth class.
- Best effort.

Each link between our backbone routers is bought from an ATM connectivity provider. We also ask for provision of ABR (available bitrate) capacity on the virtual channels. The minimal cell rate is calculated by adding the bitrate needed for IP telephony traffic

(calculated as above) to the bitrate that we want to have available for the committed bandwidth class. The maximal cell rate depends on how much bandwidth we want to offer for the best effort class.

If our provider does not have ABR, we can use CBR (constant bitrate) mode for the aggregate capacity of the real-time and committed bandwidth class, plus the spare capacity that we always want to have for best effort traffic.

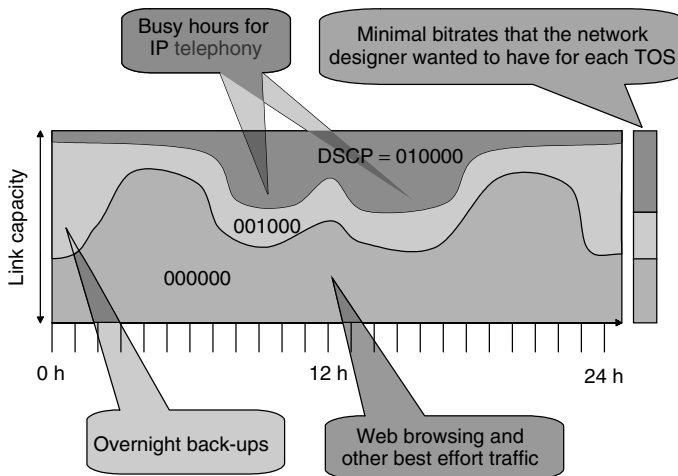
Then we configure our ingress routers to assign a DSCP codepoint = 010000 to all flows generated by IP telephony gateways (e.g., through a route-map) and TOS = 1 to all flows that need a committed bandwidth. The ingress router for such flows must also be configured to check that the bandwidth never exceeds what was reserved, in which case out-of-profile packets are marked as ‘best effort’ traffic (DSCP 000000).

If class-based queuing (CBQ) is available on our routers, we configure the queue for DSCP 010000 to always get priority treatment over 001000 and 000000. Similarly, we configure the queue for DSCP 010000 to get priority over 000000. With this configuration, IP voice packets between our gateways will never have to wait behind non-real-time traffic and committed bandwidth traffic will push back best effort traffic.

The profile of the network load on our link will look like Figure 5.15. On public backbones, this traffic mixing happens to be very favorable: business users will use a lot of bandwidth between 8 a.m. and 6 p.m., but this bandwidth will be freed for the ‘web rush’ of residential users between 8 p.m. and 11 p.m. During the empty hours (12 p.m.–6 a.m.), the provider will be able to offer special pricing for bulk data transfers.

### 5.3.2 Impact on end-to-end delay

Mixing traffic of different types does have an impact on end-to-end delay. This is because a router needs to wait until it has finished sending a packet before it can service the next



**Figure 5.15** Higher priority classes ‘eat’ the best effort budget when necessary.

one. Even if the next packet is a high-priority one, it will have to wait. On backbones dedicated to IP telephony, all packets will be small (unless techniques such as RTP-mux are used) and therefore this waiting delay will be less than on a backbone on which most packets are 1,500 octets long.

This is only worth considering on low-bandwidth links: waiting time quickly becomes negligible as the link bandwidth increases (1,500 octets are sent in 1.2 ms on 10-Mbps links). Moreover, on backbones carrying many more data flows than voice flows, the links used will have a much higher bandwidth than on a backbone dimensioned only for voice. If voice is prioritized, the delays offered on such a backbone may well be lower than those that would be observed on a dedicated voice backbone!

Clearly the trend is for ever-increasing data traffic, especially as voice traffic is limited to 24 hours per day per person! Letting voice take advantage of the high-capacity links of the data backbone is the way forward for future networks.

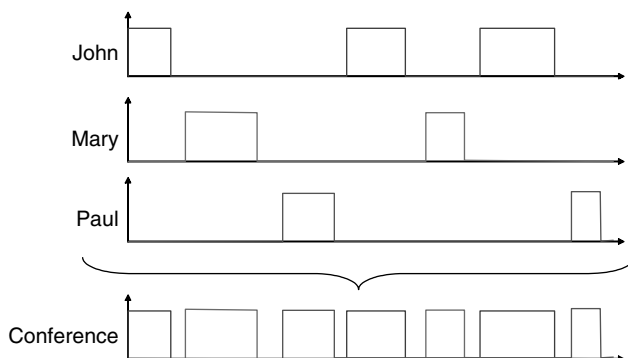
## 5.4 Multipoint communications

### 5.4.1 Audio multipoint conferences

#### 5.4.1.1 *Star topology with centralized flow mixing*

In this section we discuss the typical topology of a conference in multi-unicast mode with a central MCU. For a conference, activity rate  $a$  is calculated by considering the conference active when at least one person speaks (Figure 5.16). Parameter  $a$  will typically be in the 0.7–0.9 range, depending on whether people are bored or overexcited during the conference!

In this topology each link between the mixing server and each terminal carries asymmetric flows. If there are  $P$  participants in the conference, each speaking in turn and for the same amount of time:



**Figure 5.16** Activity rate of a multiparty conversation.



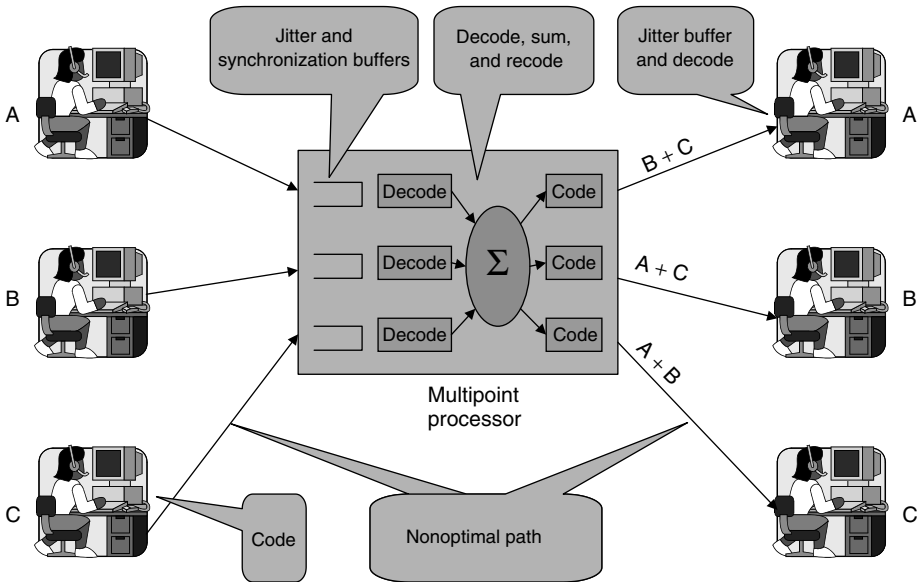
- From the server to each terminal the average traffic will be  $aM + (1 - a)m$ , with a peak value of  $M$ .
- From each terminal to the server, the average traffic will be  $(a/P)M + (1 - a/P)m$ , with a peak value of  $M$ .

The problem with multi-unicast conferences is obvious: if many participants (say,  $N$ ) are behind the same link, then average downstream traffic from the server is  $N \cdot (aM + (1 - a)m)$ . Moreover, each flow from the server to a terminal is strongly correlated and, therefore, the peak bitrate  $N \cdot M$  will be reached most of the time.

Another major issue, even if there is plenty of bandwidth, is delay. The central server must decode the audio signal it receives from each terminal, calculate the sum of all signals except the signal from the participant himself, and then recode those signals and return them. Before it can sum the signals, the central server must receive enough voice frames from each participant and synchronize them: this requires a jitter and synchronization buffer that will add at least the duration of the longest voice frame to reception delay. All other delays depend on the CPU power available, but can generally be assumed to have a duration of 2 voice frames.

Another source of delay is the nonoptimal route of voice flows, which must pass through the central server instead of reaching the receiving terminal via the shortest route. Finally, the receiving terminal still has a jitter buffer. Figure 5.17 illustrates these delays.

We conclude that multi-unicast conferences work best when installed on non-bandwidth-constrained networks and the multipoint processor is inserted in the optimal path



**Figure 5.17** Multipoint processors add significant audio delay. For clarity all terminals are represented twice: once in their sending role, once in their receiving role.

between the participants. Codecs should be used in low-delay mode (i.e., minimal number of frames per packet as allowed by bandwidth constraints).

#### 5.4.1.2 *Star topology with flow switching*

In this configuration the central server does not sum signals, but rather transmits to all participants the signal that was last measured ‘most active’. Traffic evaluation is the same as before, but delays will be lower since the coding stage needed to compress the sum signal is not needed (the decoding stage is generally still needed to evaluate signal energy, unless silence detection is used by senders). This technique allows building very simple and scalable multipoint processors.

However, this comes at a price: the conference becomes very sensitive to parasitic noise sent by senders and conference participants need to be disciplined, since it is generally impossible to interrupt an active speaker.

It is possible to improve things by building multipoint servers that mix only the last two or three most active signals: they remain scalable and yet make it possible to interrupt the active speaker. Most commercial bridges use such a compromise.

#### 5.4.1.3 *Using multicast with source-based trees*

This is the technique used for conferences on the mBone, the multicast backbone of the Internet. With multicast traffic, measurements depend a lot on the network topology between conference participants (we will examine a few typical situations below). In all cases, end-to-end delay is improved, since all flows generally follow the shortest path and the decode/recode stage is avoided.

##### 5.4.1.3.1 *Conferences over an Ethernet LAN*

Now, with the same assumptions and definitions as before, each participant generates average traffic of  $(a/P)M + (1 - a/P)m$  with a peak bitrate of  $M$ . The traffic generated by  $P$  participants is superposed on the network, and the average aggregate bitrate is therefore  $aM + (P - a)m$ , not much higher than the value obtained for a *single conversation*. If the value of  $a$  for the conference was the same as for a two-party conversation and the coder was sending no data during silences ( $m = 0$ ), then it would be equal. Note that many coders have a discontinuous transmission feature (DTX), and so we can actually have  $m = 0$  during long silences.

However, the possible peak value of aggregate traffic is  $P \cdot M$  and is obtained if all voice flows are correlated or if everybody talks simultaneously. Put another way, it is a bad idea to start a karaoke club on the Internet using multicast technology based on source-rooted trees! For normal voice conversations, however, the probability of having such a situation is very low. It is possible to limit the peak rate to  $K \cdot M$  deterministically by using terminals that refuse to send data if more than  $K$  different SSRCs are detected in incoming RTP flows (or  $K$  different source IP addresses).

The previous calculation is valid for coax or thin-coax Ethernet. However, before using a hub or a switch, *always check how they handle multicast traffic*. People usually buy

switches to prevent traffic generated by two hosts from being communicated to other hosts. Unfortunately, some older or cheap switches will turn multicast traffic into broadcast. Although this may work and your multicast conference may be fine, all other connected hosts will also receive the aggregate traffic of the conference. I've heard of networks with thousands of machines suddenly becoming desperately slow just because three guys were having a multicast videoconference!

Switches should have a way of taking into account multicast group membership information. Protocols such as CGMP (Cisco) or GARP can be used; another solution is to use switch/routers.

#### 5.4.1.3.2 LANs connected to a central router

In this topology, multicast packets are duplicated by the central router and sent to all terminals on other LANs. As before, the average traffic of each terminal to the central router is  $(a/P)M + (1 - a/P)m$  with a peak bitrate of  $M$ .

If there are  $K$  participants on a LAN, then the average traffic from the central router to this LAN will be  $(P - K)[(a/P)M + (1 - a/P)m]$ . This will be added to the traffic that is generated locally on the LAN, so the aggregate traffic is still  $aM + (P - a)m$ . The same remarks as before apply to the peak bitrate.

#### 5.4.1.3.3 Larger networks

On larger networks each source sends  $(a/P)M + (1 - a/P)m$  to a tree that reaches every participant except the source. The tree duplicates traffic according to the topology (for DVMRP, duplication is done, if there are several possibilities, where it leads to the shortest source-to-destination delays).

Comparing this with the multi-unicast situation depends on the topology of the multicast tree built by the multicast protocol. If multicast packets are duplicated very close to final users, then network load is minimized. The issue is that many multicast protocols are optimized to minimize delays and packets are duplicated at the node that minimizes the number of hops to the destination. This arguably is not the best way of minimizing traffic.

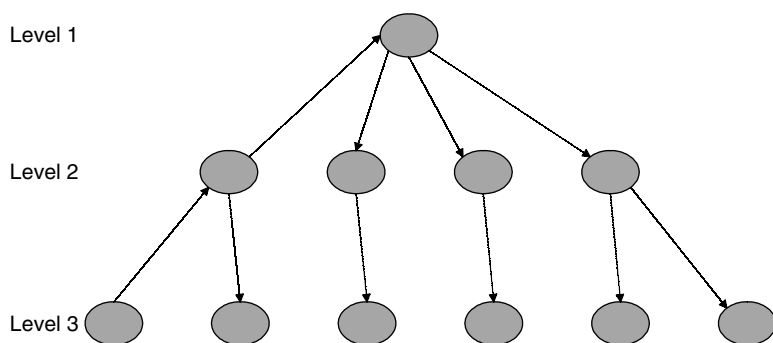
However, in some cases the shortest delay solution coincides with the lowest network load solution; that is, where there is only one possible distribution tree: in hierarchical networks (Figure 5.18).

If  $N(L)$  is the number of participants located on the part of the distribution tree rooted at link  $L$ , then we will have on this link:

- Average downstream traffic of  $(P - N(L))[(a/P)M + (1 - a/P)m]$ .
- Average upstream traffic of  $N(L)[(a/P)M + (1 - a/P)m]$ .

If we had been using a multi-unicast server at level  $i$ , the average value of downstream traffic would have been:

$$N(L)[aM + (1 - a)m]$$



**Figure 5.18** Sample hierarchical network. The shortest delay tree is also optimal for lowest bandwidth usage.

and the average value of upstream traffic:

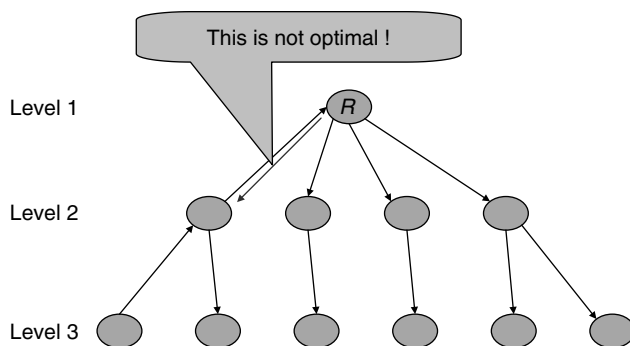
$$N(L) * [(a/P)M + (1 - a/P)m]$$

From these evaluations it is clear that the multicast solution scales better for downstream traffic if there are many participants behind the link and  $m \ll M$ . For upstream traffic the scalability of both solutions is identical.

#### 5.4.1.4 Using a shared tree multicast technology

In this technique all traffic is sent to a node (this node can be different for each conference), and multicast diffusion starts at this node (Figure 5.19). In this case, for each link  $L$ :

- Average downstream traffic is  $P * [(a/P)M + (1 - a/P)m]$ .
- Average upstream traffic is  $N(L) * [(a/P)M + (1 - a/P)m]$ .



**Figure 5.19** Shared tree multicast distribution. All sources send their flows to node  $R$ , which initiates multicast traffic.

We see that, due to the fact that local sources are heard through the central node, the result is not as good as before in terms of bandwidth. The delays are not optimal either. However, it remains better than multi-unicast if there are many participants behind that link and  $m \ll M$ .

#### **5.4.1.5 Using a hybrid unicast-multicast technique**

This technique allows every source to send their flows in unicast to a central server, but the central server is allowed to use multicast. The central server can voice-switch and redistribute the most active flow in multicast mode: although this is the most scalable design, the speaker cannot be interrupted. Another solution is to redistribute each source in multicast mode, without any mixing; this design has the same scaling properties as the shared tree multicast technique of Section 5.4.1.4, except that the number of retransmitted streams can be limited to the  $K$  most active speakers.

Note that it is impossible to perform mixing at the central node and redistribute the mixed flow in multicast to all conference participants, since the active speaker would hear his own voice echoed by the server. However, when there are a lot of passive participants (e.g., in a ‘panel’ type of conference where there are few speakers and a large audience), this mixing option is very attractive for the audience, while speakers can still use multi-unicast conferencing. This hybrid mode is used, for instance, in H.332 conferences.

#### **5.4.1.6 Conclusion**

Using multi-unicast servers for multipoint conferences is only possible for small-scale conferences. This model can be extended to the ‘panel’ service by sending the mixed audio signal to all passive members using multicast.

For larger scale conferences, the cheapest solution is to use audio switching with multicast, but this provides interactivity that is limited to a few active speakers.

High-quality conferences with many speakers can only be provided efficiently using multicast technology. There is a small difference in the bandwidth-scaling properties of shared tree and source-rooted trees, but not significant enough to be a good reason to choose one or another. If delay is a concern, source-rooted trees are a better option; but, overall the major factor of decision will be the scaling properties of such networks regarding the number of simultaneous groups that can be supported by routers (see Chapter 6 on multicast routing for more details).

### **5.4.2 Multipoint videoconferencing**

There is one major difference between audio and video conferences: video data grow linearly with the number of participants, whereas audio data are self-limiting (not everybody talks at the same time!).

In order to limit the amount of video data, most conferencing systems use video switching (i.e., broadcasting the image of the last active speaker only). Another option is to build a composite image ('mosaic') from all the incoming video streams. In this case we reach the same conclusions as for audio traffic:

- If multi-unicast is used, then traffic on each link grows linearly with the number of participants connected via this link, which rapidly leads to scalability problems.
- If the central switching server broadcasts the selected image using multicast, then the conference can grow to accommodate a larger number of active participants (sending video data), but is not limited in the number of passive participants (receivers only).

For high-quality 'continuous presence' conferences, the best solution is to have every participant send their video data to all others using multicast. Network usage is proportional to the number of simultaneous active senders.

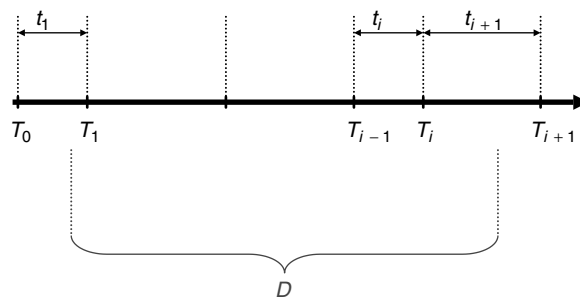
## 5.5 Modeling call seizures

### 5.5.1 Model of call arrivals: the Poisson process

The Poisson process is also called the 'memoryless' process. It is used to model many situations where the probability of future events does not depend on past events. Put another way, when knowledge of past events tells us nothing about the future (e.g., the Poisson process can be used to model the disintegration of radioactive elements). Under certain conditions it can be used to model the occurrence of incoming calls.

Let's number each new call set-up as 'event number  $i$ ' and its timestamp  $T_i$ . The time interval between event  $i - 1$  and event  $i$  is  $t_i$  (Figure 5.20).

If the probability of having a new call between  $t$  and  $t + dt$  does not depend on previous events and is proportional to  $dt$  (first-order approximation) with a negligible probability that more than one call arrives when  $dt$  is small, then it is a Poisson arrival process.



**Figure 5.20** The Poisson call arrival model.

If we call  $P_k(D)$  the probability of having exactly  $k$  calls during a time interval  $D$ , then our assumptions can be written:

$$P_1(d) = \lambda \times d + o(d)$$

$$1 - P_0(d) - P_1(d) = o(d)$$

$$P_{k+1}(D + d) = P_{k+1}(D)(1 - \lambda d) + P_k(D) \times \lambda d + o(d)$$

where the last line is obtained because we can have  $k + 1$  events at  $D + d$  if we have:

(a)  $k + 1$  events at  $D$  and no event between  $D$  and  $D + d$ .

or

(b)  $k$  events at  $D$  and one event during  $d$ .

This leads to a differential equation whose solution is:

$$P_k(D) = \frac{(\lambda D)^k}{k!} e^{-\lambda D}$$

and if we call  $A(t)$  the probability that  $t_1 < t$ , we have:

$$A(t) = 1 - P(t_1 > t) = 1 - e^{-\lambda t}$$

The probability of not having any event during  $t$  is purely exponential, and the Poisson distribution is often called the ‘exponential distribution’:

$$B(t) = 1 - A(t) = e^{-\lambda t}$$

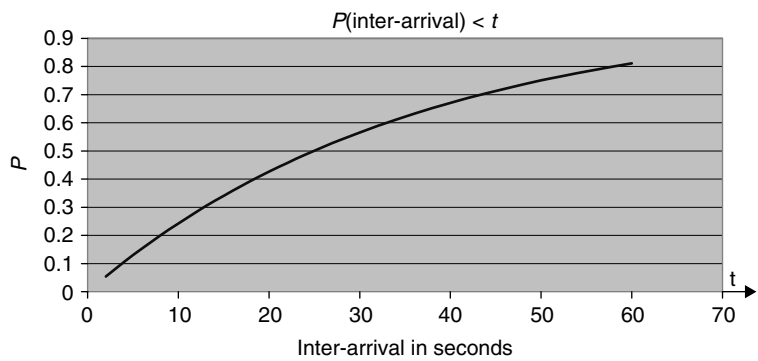
In fact,  $\lambda$  is *the average frequency of the event*, and  $1/\lambda$  is *the mean inter-arrival time* ( $\bar{t} = \int t A'(t) dt = 1/\lambda$ ).

Incoming phone calls are often modeled as a Poisson process in large networks, even if one could argue that our hypothesis that an event does not depend on past events is not always true. This is because the superposition of  $N$  independent processes  $P_i$  with an average event frequency  $\lambda_i$ , even if they are not individually Poisson processes, converges to a Poisson process when  $N$  increases (Palm–Kintchine theorem). The average frequency of this Poisson process is the sum of all  $\lambda_i$ . This is exactly what happens for a set of  $N$  telephone lines. If  $f$  is the frequency of calls for each line,  $A$  the total traffic on the network, and  $d$  the average duration of a call, call set-up events can be considered as a Poisson process where  $\lambda = Nf = A/d$ . Figure 5.21 shows the inter-arrival distribution obtained for 100 lines and 1 call/hour per line.

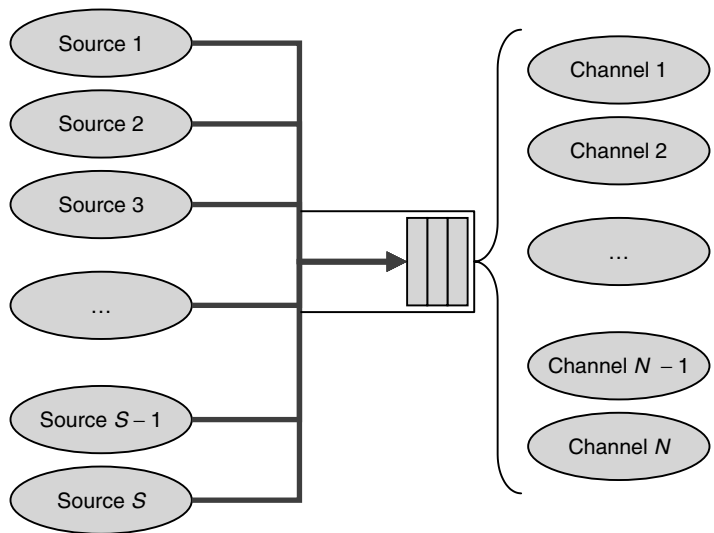
## 5.5.2 Model of a call server

### 5.5.2.1 Queue model

The problem of arriving calls that are served, queued, or rejected by a telephone switch is an example of a queuing problem. A queue is modeled as a number  $S$  of ‘sources’ (finite or



**Figure 5.21** Probability of interarrival period to be smaller than  $t$ .



**Figure 5.22** Generic queue model.

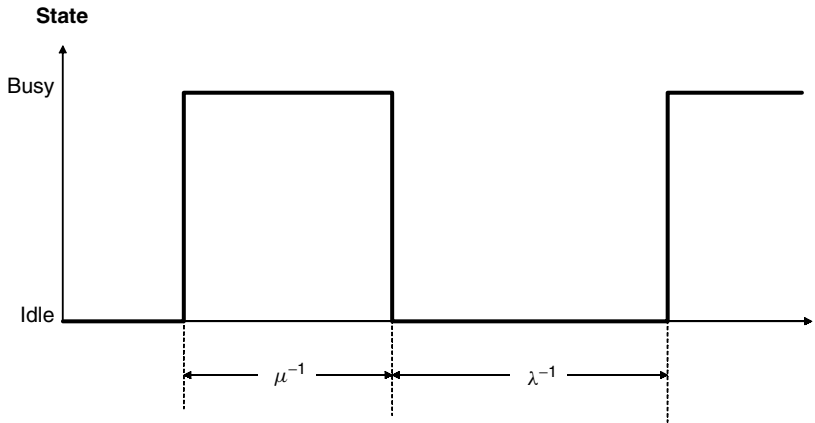
infinite) which generate requests processed by a number  $N$  of ‘channels’ (Figure 5.22). These requests can be served immediately if there is an available channel, queued if allowed by the system, or rejected if the queue buffer is full or does not exist.

Each source is modeled as a two-state process: the ‘idle’ interval is exponentially distributed with intensity  $\lambda$ , and the busy interval is exponentially distributed with intensity  $\mu$  (as shown in Figure 5.23).

**5.5.2.2 Different assumptions lead to various models**

The various models listed in Table 5.9 all derive from the same queue model, but with different assumptions. Each model is described in the following sections.





**Figure 5.23** Two-state exponential source model.

**Table 5.9** Model to use according to traffic assumptions

Call arrival model	Independent sources, two-state model, each with an exponential distribution				
Call arrival intensity	Varies proportionally to the number of on-hook lines		Fixed (variations according to the number of on-hook lines are neglected)		
Congestion	No congestion: full availability $N = S$	Congestion: $N > S$	No congestion: full availability $N = S$	Congestion: $N > S$	
Congested calls	None	Lost	None	Lost	Queued
Service model	Any distribution				Poisson + FIFO
<i>Model</i>	<i>Bernoulli</i>	<i>Engset</i>	<i>Poisson</i>	<i>Erlang B</i>	<i>Erlang C</i>

Neglecting variation in arrival intensity according to the number of lines already off-hook (which cannot generate calls) amounts to considering incoming traffic as a pure Poisson process, whose intensity does not depend on the past history of the system. It is valid only if each line is ‘on-hook’ for a small fraction of the time, which is generally true, and if there are enough lines so that the influence of an off-hook line can be neglected.

The Bernoulli and Engset models do not consider arriving traffic globally as a fixed intensity Poisson process, but instead consider the probability of having a new call seizure for each on-hook line as a Poisson process.

### 5.5.3 Dimensioning call servers in small networks

#### 5.5.3.1 Overdimensionned network: Bernoulli distribution

This is a special case where there are as many channels as sources ( $N = S$ ). It is also called a 'full availability system', because no client can be rejected or even wait.

We consider many ( $G$ ) similar environments with  $S$  sources and  $N$  channels. If we take a snapshot at any time, some of those systems will have only one line busy, some of them will have two lines busy, and some up to  $S$  lines busy. We call  $g(n)$  the number of systems with  $n$  busy lines (' $n$ ' systems). We assume that if the group of systems is large, the number  $g(n)$  of systems in each state ' $n$ ' is steady.

We consider the evolution of these  $G$  systems over a very short time interval  $t$ , so short that there is a negligible probability of having more than one event, like a new call, or completion of an active call for each system: the probability of transiting from state ' $n$ ' to state ' $n + 2$ ' or ' $n - 2$ ' for each system is negligible.

Let's calculate the number of new calls received during a very short interval  $t$  by  $g(n)$  ' $n$ ' systems, turning them into ' $n + 1$ ' systems. As discussed above, thus is  $g(n) * (S - n) \lambda t$ . Similarly, calls will be completed in ' $n + 1$ ' systems, turning them into ' $n$ ' systems. In our brief interval  $t$ , there will be  $g(n + 1) * (n + 1) * \mu t$  such occurrences.

The distribution of systems in each state is steady only if the flow of transitions in both directions is equal (Figure 5.24).

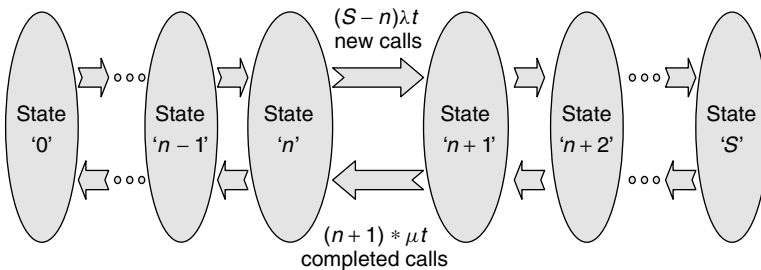
If we want distribution  $g(n)$  to be steady, the number of ' $n$ ' systems becoming ' $n + 1$ ' systems must equal the number of ' $n + 1$ ' systems becoming ' $n$ ' systems during  $t$ ; therefore, we must have for each  $n$ :

$$0 = (n + 1) \mu t * g(n + 1) - g(n) * (S - n) \lambda t$$

In fact,  $g(n)/G$  is the probability  $P(n)$  of having  $n$  lines simultaneously busy and the exact value of  $t$  has no importance. Our equation becomes:

$$P(n + 1) = P(n) * (S - n) * \lambda / \mu(n + 1) \quad (\text{eq. } n+1)$$

$$P(n) = P(n - 1) * (S - n + 1) * \lambda / \mu(n) \quad (\text{eq. } n)$$



**Figure 5.24** Transition model for the Bernoulli distribution.

This leads by recurrence to:

$$P(n) = \frac{S!}{n!(S-n)!} \left(\frac{\lambda}{\mu}\right)^n P(0) = C_S^n \times \left(\frac{\lambda}{\mu}\right)^n P(0)$$

Since the sum of  $P(n)$  over all values of  $n$  equals 1, and we have:

$$\sum_{n=0}^N P(n) = \left(\frac{\lambda}{\mu} + 1\right)^S P(0)$$

we can find:

$$P(0) = \left(\frac{\mu}{\lambda + \mu}\right)^S = \left(\frac{1}{1 + \beta}\right)^S$$

where  $\beta = \lambda/\mu$  is called the offered traffic per idle source:

$$\begin{aligned} P(n) &= C_S^n (\beta)^n \left(\frac{1}{1 + \beta}\right)^S = C_S^n \left(\frac{\beta}{1 + \beta}\right)^n \left(\frac{1}{1 + \beta}\right)^{S-n} \\ &= C_S^n \left(\frac{\beta}{1 + \beta}\right)^n \left(1 - \frac{\beta}{1 + \beta}\right)^{S-n} \end{aligned}$$

In the case where calls are never blocked, traffic  $a$  per source equals  $\lambda/(\lambda + \mu)$  or  $\beta/(1 + \beta)$  and we have:

$$P(n) = C_S^n \times a^n \times (1 - a)^{S-n}$$

Yes, this is the same binomial (Bernoulli) distribution as the one we encountered in the first section 5.1.2 when trying to calculate the probability of having  $n$  simultaneous active voice channels. If we consider calling lines as either ‘active’ with probability  $a$  or ‘idle’ with probability  $1 - a$ , and calculate the probability of having  $n$  from  $S$  lines active, then  $a$  denotes the proportion of time the line is busy, not the proportion of time it is sending voice packets when active.

The standard deviation of this distribution is  $\sigma = \sqrt{Sa(1 - a)}$  (note that it varies proportionally to the square root of total traffic  $A = Sa$ ). Therefore, it is more efficient to process calls from a large number of lines: a large network-based Centrex switch serving 1,000 small enterprises will require a lot less simultaneous call capacity than the sum of 1,000 small IP PBXs! This will also be reflected in the results of Section 5.5.3.2.

### 5.5.3.2 The PBX model: $X$ clients and $N$ servers (the Engset distribution)

The full availability model is of little use in practice, because we want to use the smallest set of servers (PBX capacity) for offered traffic. We accept losing some calls if congestion occurs, but only a small fraction. We use the same definitions as above:

- $S$  is number of sources.
- $N$  is number of channels (now we have  $N < S$ ).

- $\lambda$  is call intensity per idle source.
- $1/\mu$  is mean holding time.
- $\beta = \lambda/\mu$ .

In addition, let's call  $B$  the probability that a new call gets rejected (i.e., that  $N$  servers are used when the call arrives).

We calculate the new distribution  $P(n)$  using the same method, by considering state ' $n$ ' to ' $n + 1$ ' transitions as in equilibrium. When  $n \leq S$  the equation set is the same as above:

$$P(n) = P(n-1) * (S - n + 1) * \lambda / \mu(n) \quad (\text{eq. } n)$$

Therefore, we also have:

$$P(n) = C_S^n \times \left(\frac{\lambda}{\mu}\right)^n P(0)$$

We must have:

$$\sum_{n=0}^N P(n) = 1$$

which leads to:

$$P(n) = \frac{C_S^n * \left(\frac{\lambda}{\mu}\right)^n}{\sum_{i=0}^N C_S^i * \left(\frac{\lambda}{\mu}\right)^i} = \frac{C_S^n * (\beta)^n}{\sum_{i=0}^N C_S^i * (\beta)^i} = E_{n,S}$$

This is simply the truncated form of the binomial distribution. In order to avoid computer overflows when evaluating the Engset distribution, the most efficient method is to use the following recursion formula:

$$\frac{1}{E_{i,S}(\beta)} = 1 + \frac{i}{\beta \cdot (S - i + 1)} \cdot \frac{1}{E_{i-1,S}(\beta)}$$

$$E_{0,S}(\beta) = 1$$

The time congestion probability (the proportion of time the system is blocked for new call attempts) is  $P(S)$ . The probability of call congestion  $B$  is smaller than the time probability of having  $S$  servers occupied. This is because average arriving traffic intensity gets smaller as the number of busy servers increases (i.e., fewer on-hook lines, see the  $S - n$  coefficient in equilibrium equations), and therefore the arriving intensity in state  $n = S$  is below average.

Call attempts arriving in state  $n = S$  are blocked:

$$B_{N,S} = \frac{\text{Lost\_calls}}{\text{Total\_call\_attempts}} = \frac{P(N) \cdot (S - N) \cdot \lambda}{\sum_{i=0}^N P(i) \cdot (S - i) \cdot \lambda}$$

$$= \frac{C_S^N \cdot \beta^N \cdot (S - N) \cdot \lambda}{\sum_{i=0}^N C_S^i \cdot \beta^i \cdot (S - i) \cdot \lambda} = \frac{C_{S-1}^N \cdot \beta^N}{\sum_{i=0}^N C_{S-1}^i \cdot \beta^i} = E_{N,S-1}$$

In other words, the probability that a new call attempt gets rejected equals the probability that remaining  $S - 1$  sources fully occupy the  $N$  channels. This result, called the ‘arrival theorem’ is valid for any queuing system with a limited number of sources, even if congested calls are lost or delayed.

There is the following relation between  $E_{N,S}$  and  $B_{N,S}$ :

$$E_{N,S} = \frac{S}{S - N} \cdot \frac{B_{N,S}}{1 + \beta(1 - B_{N,S})}$$

Carried traffic  $A_c = S \cdot a_c$  can be evaluated by substituting  $P(i)$  by its expression in the equilibrium equation:

$$\begin{aligned} A_c &= \sum_{i=1}^N i \cdot P(i) = \sum_{i=1}^N \beta \cdot (S - i + 1) \cdot P(i - 1) \\ &= \sum_{i=0}^{N-1} \beta \cdot (S - i) \cdot P(i) = \beta S(1 - P(N)) - \beta A_c + \beta N \cdot P(N) \\ A_c &= S a_c = \frac{\beta}{1 + \beta} \cdot [S - E_{N,S} \cdot (S - N)] = S - \frac{S}{1 + \beta \cdot (1 - B_{N,S})} \\ a_c &= 1 - \frac{1}{1 + \beta \cdot (1 - B_{N,S})} \end{aligned}$$

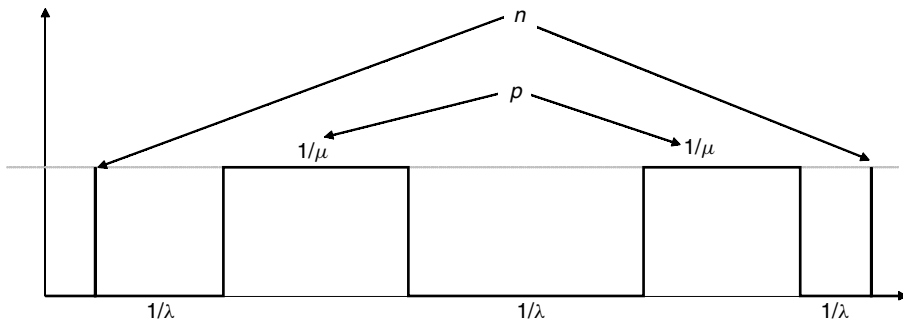
Usually, the problem is to find the blocking probability for a given number of servers  $N$  when offered traffic  $A_o$  is known, not when  $\beta$  is known. Here the definition of offered traffic varies from author to author. Some authors simply define offered traffic  $A$  as  $A = A_c/(1 - B)$  or the equivalent  $(A - A_c)/A = B$ . In this case there is no easy analytical solution.

For very small values of  $B$ , it is possible to approximate  $(1 - B)$  by 1. Another way of solving the problem is to draw  $(A, B)$  parametrically as a function of  $\beta$  for the given values  $B$  and  $N$ . The desired result is the value of  $B$  when  $A = A_o$ . This definition of offered traffic is used by most traffic calculators available on the Internet.

The ITU question 16 teletraffic group defines offered traffic as the traffic that would be carried in the total accessibility situation (no blocking). This is more logical, as with this definition offered traffic does not depend on the number of servers. With this more rigorous definition we have:

$$A = \frac{\lambda}{\lambda + \mu} = \frac{\beta}{1 + \beta}$$

With this definition it is easy to find  $\beta$  from  $A$ , and derive the time-blocking probability  $P(N)$  and the call-blocking probability. With this definition, however, we no longer have



**Figure 5.25** Traffic profile when some calls are rejected.

$(A - A_c)/A = B$ , but instead:

$$C = \frac{A - A_c}{A} = \frac{B}{1 + \beta(1 - B)}$$

where  $C$  is called traffic congestion (representing approximately the loss of revenue for a service provider). While this difference seems counter-intuitive, it can be understood by considering real traffic with lost calls for a source (Figure 5.25).

The number of active periods is  $p$ , the number of failed call attempts is  $n$ , and we have  $B = n/(n + p)$ . Over a period of time  $T$ , the line transmits  $p/\mu$  seconds of traffic. Therefore, we have:

$$a_c = \frac{P/\mu}{T} = \frac{P/\mu}{p(1/\mu + 1/\lambda) + n/\lambda} = \frac{\beta p}{p(1 + \beta) + n} = \frac{\beta(1 - B)}{(1 - B)(1 + \beta) + B}$$

$$C = \frac{a - a_c}{a} = 1 - a_c \cdot \frac{\beta + 1}{\beta} = \frac{B}{1 + \beta(1 - B)}$$

If we consider  $C = B$ , we are overlooking the fact that when a call cannot complete it does not spend any time active, and therefore on average the next call arrives after  $1/\lambda$ , not after  $1/\lambda + 1/\mu$ , which explains why we have  $C < B$ .

## 5.5.4 Dimensioning call servers in large networks

### 5.5.4.1 Full availability model

In a large network, the total number of end points will usually be unknown; only the total traffic  $A$  to be processed will be available. Although  $S$  is very large and unknown, and  $a$  is small and unknown, we know that  $S \cdot a = A$ , the total traffic (in Erlangs) generated by the sources. Therefore, we cannot use the Bernoulli distribution as presented above, but

the Bernoulli distribution  $P(n)$  can be simplified<sup>3</sup> into a Poisson law:

$$P(n) = \frac{A^n}{n!} e^{-A}$$

and we approximate  $\sigma = \sqrt{A}$ , which is slightly larger than the exact value.

Note that the average of both the Bernoulli and the Poisson laws is obviously  $A$  (the amount of arriving traffic, since there is no congestion). However, the variance of the Poisson law is higher than the variance of the Bernoulli distribution: so when using the Poisson law to dimension a network with a large, unknown number of clients, we obtain a worst case estimate.

#### 5.5.4.2 The Erlang B formula

We can derive the Erlang B law from the Engset distribution if each line is not very active, there are many lines, and we only know the total traffic ( $Sa \rightarrow A$ ). In this approximation the variation of incoming traffic intensity with the number of off-hook lines can be ignored ('infinite' number of sources) and the equilibrium equation converges to:

$$P(n) = Sa/n * P(n-1)$$

and we have:

$$P(n) = \frac{\frac{A^n}{n!}}{\sum_{i=0}^N \frac{A^i}{i!}}$$

This is a truncated Poisson distribution. The probability of call loss is:

$$B = P(S) = E_{1,S}(A)$$

This result is the first Erlang law (also called the Erlang B law). It allows calculation of the number  $N$  of servers (e.g., external lines) that must be installed to ensure a given rejection rate  $B$ , if the excess calls are *immediately* rejected. This is similar to the Poisson law, with a scaling factor.

Since the Erlang law is obtained simply by considering that arrival intensity does not depend on the number of clients already 'off-hook', time congestion probability  $P(S)$  is now equal to call loss probability  $B$ .

The Erlang B law is valid only if the law of arrival calls is a Poisson process and if congested calls are cleared, but makes no assumption on call duration distribution, which can be exponentially distributed (Poisson law) or fixed.

---

<sup>3</sup> Note that this limit of the binomial distribution is different from that of the Moivre approximation used in Section 1.2: it is obtained when  $S$  increases, but with a constant value for  $a$ .

In order to compute the Erlang B law without overflow errors, the following recursive method can be used:

$$\frac{1}{E_{1,N}(A)} = 1 + \frac{N}{A} * \frac{1}{E_{1,N-1}(A)} \quad \text{with } E_{1,0}(A) = 1$$

It is always possible to compute  $P(N)$  with increasing values of  $N$  until we reach the desired  $B$ , but there is also a simple approximation that is useful to make quick order-of-magnitude estimations (known as the ‘Rigault rule’): if the desired  $B$  is  $10^{-k}$ , then  $N_{\max} \approx A + k\sqrt{A}$ .

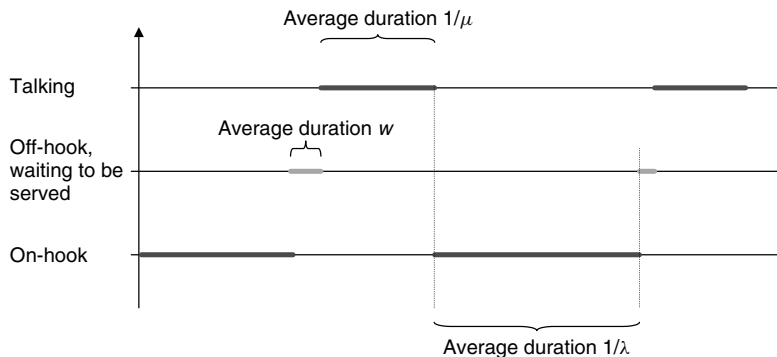
#### 5.5.4.3 Model for a limited set of servers and phone lines that can wait for a server (Erlang C formula)

A call server that can process a maximum of  $N$  calls simultaneously, instead of dropping all calls that arrive while the servers are busy, can decide to queue the incoming calls for a little while. This model can also be used for call centers.

Let’s now model the traffic generated by a single phone line with an average pick-up frequency per line of  $\lambda$ , an average duration  $1/\mu$ , and an average waiting time of  $w$ , as illustrated in Figure 5.26.

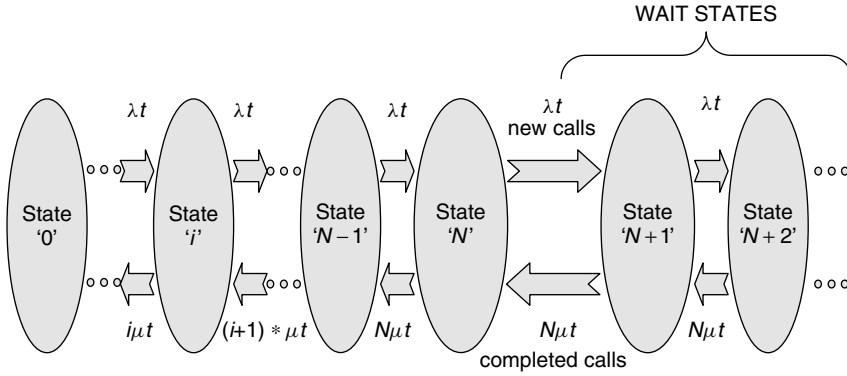
A line is served immediately it is picked up, if possible, but if there are already  $N$  lines busy then the user needs to wait to be served. The waiting queue size is infinite and the users are served in FIFO mode.

The method used to obtain the probability of each state is similar to that used previously. In addition to the  $N$  states, corresponding to the number of channels used, there are additional states corresponding to all  $N$  channels used, and 1, 2, 3, ... lines waiting for service in the queue. For these states we use the notation ‘ $N + 1$ ’, ‘ $N + 2$ ’, ‘ $N + 3$ ’, etc. (Figure 5.27). In all waiting states only  $N$  channels process calls and, therefore, call-processing intensity remains fixed at  $N\mu t$ .



**Figure 5.26** Call states for the Erlang C model.





**Figure 5.27** State transitions for the Erlang C model.

Under the same approximation conditions as for the Erlang B law (i.e., many lines served, each with relatively little traffic), the offered traffic is  $A = \lambda/\mu$  and we obtain:

$$P(j < N) = \frac{A^j}{j!} P(0) = \frac{\frac{A^j}{j!}}{\left(1 + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots + \frac{A^N}{N!}\right) + \frac{A^N}{N!} \times \left(\frac{A}{N-A}\right)}$$

$$P(N+j) = \left(\frac{A}{N}\right)^j P(N) = \frac{\left(\frac{A}{N}\right)^j \cdot \frac{A^N}{N!}}{\left(1 + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots + \frac{A^N}{N!}\right) + \frac{A^N}{N!} \times \left(\frac{A}{N-A}\right)}$$

The time probability of waiting called  $E_{2,N}(A)$  can be obtained summing all  $P(N+j)$ . It can be shown that in the case of a Poisson arrival process this is also the call-waiting probability (the PASTA, or Poisson Arrivals See Time Averages, property). This formula is the Erlang C law:

$$E_{2,N}(A) = \sum_{j=0}^{\infty} P(N+j) = \frac{N}{N-A} P(N) = \frac{\left(\frac{N}{N-A}\right) \frac{A^N}{N!}}{1 + A + \frac{A^2}{2!} \cdots + \frac{A^N}{N!} + \frac{A^N}{N!} \left(\frac{A}{N-A}\right)}$$

It is also interesting to note that the Erlang C law can be expressed as a function of the Erlang B law:

$$E_{2,N}(A) = \frac{N E_{1,N}(A)}{(N-A) + A E_{1,N}(A)}$$

The probability of waiting more than  $t_w$  seconds can be obtained because, in each state ' $N+j$ ', this probability is the same as having fewer than  $j$  departures in  $t_w$  seconds, if

the service discipline is FIFO. For a Poisson service process, this is:

$$\sum_{r=0}^j \frac{\left(\frac{N}{\mu} t_w\right)^r}{r!} e^{-\frac{N}{\mu} t_w}$$

The weighted sum on all waiting states gives:

$$P(w > t_w) = E_{2,N}(A) \cdot e^{-\frac{(N-A)t_w}{\mu}}$$

The average number of calls waiting in the queue, also called waiting time traffic, is:

$$A_w = \sum_{j=0}^{\infty} j P(N+j) = \frac{AN}{(N-A)^2} P(N) = \frac{A}{N-A} E_{2,N}(A)$$

and Little's theorem (valid in most queuing systems) tells us that the average waiting time  $t_{wait1}$  equals the average queue length divided by the arrival intensity:

$$t_{wait1} = \frac{A}{(N-A) \cdot \lambda} E_{2,N} = \frac{1}{\mu(N-A)} E_{2,N}$$

This is an average for all calls, including those that do not wait. The average for calls that wait  $t_{wait2}$  can be obtained by writing total waiting time as:

$$\begin{aligned} Total\_wait\_time &= t_{wait1} * Total\_calls \\ &= t_{wait2} * Waiting\_calls = t_{wait2} * Total\_calls * E_{2,N}(A) \end{aligned}$$

Therefore the average waiting time for calls that effectively wait is simply:

$$t_{wait2} = \frac{1}{\mu(N-A)}$$

## 5.6 Conclusion

Systems with few users need to be dimensioned for peak values; systems with many users need to be dimensioned for average values. This allows network use to become more efficient as the number of users increases.

In a large TDM telephony system, the network can be dimensioned for slightly more than the average number of simultaneous calls. It is possible to be even more efficient with IP telephony systems, because audio coders with voice activity detection make it possible to perform some statistical multiplexing between active channels and idle channels: when there are many channels, the link needs to be dimensioned for slightly more than the average bitrate of the coder taking into account the activity rate of the conversation. Another potential advantage of VoIP is that it is able to use many low-bitrate coders, not

just the 64-kbit/s G.711 codec used in traditional telephony. Some of these low bitrate coders have a peak rate of less than 10 kbits/s.

With an additional level of multiplexing (voice activity detection) and more efficient coders, it would seem that IP telephony is far more efficient than the PSTN. In fact, this comment must be viewed with skepticism. The tremendous overheads of RTP+UDP+IP+the physical layer ‘eat’ much of what was gained with compression and VAD (many systems do not even have VAD). Overheads can be reduced at the edge of the network using compressed RTP (cRTP), but within the core network overhead reduction requires stacking many frames per RTP packet and this degrades delay performance.

Overall, it seems that claiming that *the* advantage of IP telephony is its bandwidth efficiency is misleading. Many other techniques can achieve the same or even better efficiency (e.g., think of DCME equipment used on transatlantic lines). The key advantages of VoIP lay elsewhere:

- Much better any-to-any connectivity in large networks (compared with frame relay or ATM where too many ‘virtual channels’ are required).
- Companies and service providers can reduce wiring costs, as all communications are multiplexed on the data network.
- Carriers can merge all communications on a single backbone, reducing maintenance and operations costs.
- VoIP voice-switching equipment does not need to route media streams, unlike TDM switches. Because of this they can be located much farther away from the end-customer, reducing the need for local POPs and reducing both CAPEX and ongoing OPEX.
- The commoditization of hardware used for service platforms, because the hard, real-time requirements of TDM are no longer required in VoIP (the media stream bypasses the service equipment); so, standard computers can be used instead of purpose-built TDM systems.

With the ever-increasing availability of broadband connections at an affordable price and the constant lowering of the cost of long-distance bandwidth for carriers, service providers now have a tendency to focus on services, instead of spending a lot of time optimizing the need for network resources. More and more carriers are now deploying voice over IP networks with the G.711 voice coder, because they find the gain in bandwidth is not worth the reduction in perceived voice quality. Service providers may even, in the near future, introduce broadband coders in their networks. VoIP started with a focus on cheaper voice and prepaid telephony, but it is now clear that the direction is toward the high-end market, advanced services, and a richer multimedia experience.

## 5.7 References

- R. Guerin, H. Hamadi, and M. Naghshineh (1991). Equivalent capacity and its application to bandwidth allocation. *IEEE Journal on Selected Areas in Communications*, 9(7), September.
- L. Kleinrock. *Queueing Systems*, Wiley Interscience (ISBN 0 471 49110 1).