1 Motivation of TCP Congestion Control

During the transmission in the networks, sometimes the externally offered load is larger than the actual network capacity can be handled. In such situation, packet delays will increase and the actual network throughput will decreases. So how to avoid this type of congestion is one of most important problems in networkding. Usually there are two kinds of Congestion Control:

- Open-Loop Control:
  During connection setup, the transmission rate of a connection is determined.
  - Call Admission Control: Before the connection setup, there is a "conversation" between two process and they negotiate on some "service contract", an agreement on some service parameters for the session's input traffic(maximum rate, minimum rate, maximum burst size etc.). A typical situation is that of the voice telephone network and circuit switched networks. For example, in the voice telephone service, the sender requires the minimum transmission rate is c and the receiver can provide such a service, then there is an agreement between them and the setup is connected.
  - Policing: Leaky Bucket: There is a problem in call admission control, which is how do we make sure that the connection does not transmit packet at a higher rate after the connection is setup. So a related method to regulate the burstiness of the transmitted traffic is *leaky bucket schema*. The packet in the queue can be transmitted only after getting a permit and the permits are generated at the desired input rate r of the session. By this schema, we can make sure that the transmission rate in the connection won't exceed the extent r what we want. For the details of leaky bucket, please refer to the book.
- Closed-Loop Control / Feed Back Control:
  Connections are informed dynamically about the congestion state of the network, and asked to adapted their rate accordingly.
  - TCP Congestion Control: TCP provides a reliable transport service between two processes running on different hosts. TCP congestion control regulates the transmission rate dynamically according to the congestion state of the network using the feedback from end to end.

When multiple TCP connections share the bandwidth of a link, the goal of TCP Congestion-control mechanism is to control the transmission rate so that the sum of the transmission rate of all connections is less than the capacity of the shared link. If there are R connections in the network, the transmission rate of r^{th} connection is X_r and the the capacity of the whole link is C, then their relation must satisfy the following inequality:

$$\sum_{r=1}^{R} X_r < C$$

2 TCP Congestion Control Algorithm

TCP uses ARQ to provide the reliable service between different hosts. A TCP connection controls its transmission rate by limiting its number of transmitted-but-yet-to-be-acknowledged segments. This number of permissible unacknowledged segments is often referred to as the TCP **window size**. Ideally, TCP connections should be allowed to transmit as fast as possible(that is, to have as large a number of outstanding unacknowledged segments as possible) as long as segments are not lost (dropped at routers) due to congestion. To avoid congestion, there are two kinds of controls for different purposes: Flow Control and Congestion Control.

2.1 Flow Control:

A mechanism to prevent the sender from sending data when the receiver buffer is full. This is necesary between two users for speed matching, that is, for ensuring that a faster transmitter does not overwhelm a slow receiver with more packets than the latter can handle.

In "flow control" we use $RcvWin$ to mark the upperbound that a receiver can handle. The amount of unacknowledged data that a host can within a TCP connection may not exceed $RcvWin$ in order to prevent the faster sender from overwhelming the slow receiver with more packets than can be handled.

2.2 Congestion Control:

A mechanism to prevent congestion within the network by regulating the packet population, hence the transmission rate within the subnetwork.

In "congestion control" we define another variable: the $CongWin$. This window size imposes an additional constraint on how much traffic a host can send into a connection without leading to congestion.

Next we will look at how the congestion window constraints the transmission rate:

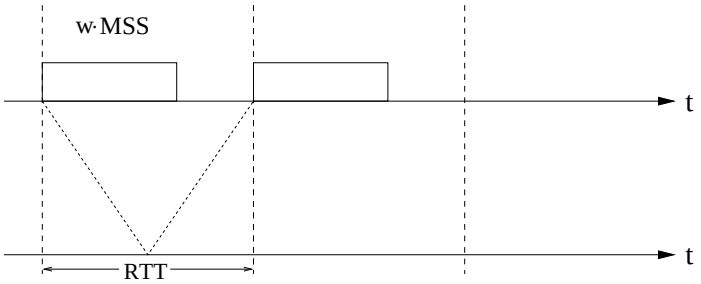

Figure 1: Congestion Window-Transmission Rate

- w: window size, that is the number of of permissible unacknowledged segments
- MSS: Maximum Segment Size, TCP encapsulates each chunk of client data with a TCP header, thereby forming TCP segments and MSS is the maximum amount of data that can be grabbed and placed in a segment.

- RTT: Round-trip time, that is the time between the staring point when the sender begin to transmit segments and the time that the sender receives acknowledgements for these segments.

If a TCP sender transmits all w segments back to back, it must wait for one round-trip time (RTT) until it receives acknowledgements for these segments, at which point it can send w additional segments. If a connection transmits w segments of size MSS bytes every RTT seconds, then the congestion window is $wMSS$ bytes and the connection's throughput, or transmission rate is $(wMSS)/RTT$ bytes per second.

$$CongWin = wMSS \qquad\qquad x(w) = \frac{wMSS}{RTT}$$

2.3 TCP Congestion Algorithm Description(Tahoe):

Parameters

- $RcvWin$
- $threshold$
- w
- $CongWin = wMSS$
- $n = \min\{RcvWin, CongWin\}$

We have explained the meaning of $RcvWin$, $CongWin$ and w before, then what is $threshold$? It is a variable that affects how $CongWin$ grows. $CongWin$ has different growing patterns below and above the $threshold$ value. We will discuss in detail below.

Initiate

- Set $threshold$
- Set $w = 1$

Congestion Control Process:

In order to focus on congestion control, we assume that the TCP $RcvWin$ is so large that its constraint can be ignored. In this case, the amount of unackknowledged data that a host can have within a TCP connection is solely limited by $CongWin$. Furthure we assume that a sender has a very large amount of data to send to a receiver.

Once a TCP connection is established between that two end systems, the application process at the sender writes bytes to the sender's TCP send buffer. TCP grabs chunks of size MSS, encapsulates each chunk within a TCP segment, and passes these segments to the network layer for transmission acorss the network. The TCP congestion window regulates the times at which the segments are send into the network(that is, passed to the network layer).

- Slow start
  - As long as $w \leq threshold$, for every received ACK set

$$w = w + 1$$

Initially, the congestion window is equal to one MSS. TCP snds the first segment into the network and waits for an acknowldgement. If this segment is acknowledged before its timer times out, the sender increases the congestion window by one MSS and send out two maximum-size segments. If these segments are acknowledged before their timeouts, the sender increases the congestion window by on MSS for each of the acknowledged segments, giving a congestion window of four MSS, and sends out four maximum-sized segments. This procedure continues as long as (1) the congestion window is below the threshold and (2) the acknowledgements arrive before their corresponding timeouts.

During this phase of the congestion-control procedure, the congestion window increases exponentionally fast. The congestion window is intialized to one MSS; after one RTT, the window is increased to two segments; after two round-trip times, the window is increased to four segments; after four round-trip times, the window is increased to eight segments, and so forth. This phase of the algorithm is called "slow start" because it begins with a small congestion window equal to one MSS. (The transmission rate of the connection starts slowly but accelerates rapidly.)

- Congestion Avoidance
  - When $w > threshold$, for every w ACK received set

$$w = w + 1$$

The slow-start phase ends when the window size exceeds the value of *threshold*. Once the congestion window is larger than the current value of *threshold*, the congestion window grows linearly rather than exponentially. Specifically, if w is the current value of the congestion window, and w is larger than *threshold*, then after w acknowledgments have arrived, TCP replaces w with $w + 1$. This has the effect of increasing the congestion window by 1 in each RTT for which an entire window's worth of acknowledgments arrives.

- Loss
  - When a ACK is not received when time-out expires, set

$$threshold = \frac{w}{2}$$

$$w = 1$$

  - Got to "slow start"
    The congestion-avoidance phase continues as long as the acknowledgments arrive before their correspnding timeouts. But the window size, and hence the rate at which the TCP sender can send, cannot increase forever. Eventually, the TCP rate will be such that one of the links along the path becomes saturated, at which point loss(and a resulting timeout at the sender) will occurs, the value of *threshold* is set to half the value of the current congestion window and the congestion window is reset to one MSS. the sender then again grows the congestion window exponentially fast using the slow-start procedure.

In summary:

- When the congestion window is below the threshold, the congestion window grows exponentially.
- When the congestion window is above the threshold, the congestion window grows linearly.
- Whenever there is a timeout, the threshold is set to one-half of the current congestion window and the congestion window is then set to 1.

If we ignore the slow-start phase, we see that TCP essentially increases its window size by 1 each RTT (and thus increase its transmission rate by an additive factor) when its network path is not congested, and decreases its window size by a factor of 2 each RTT when the path is congested. For this reason, TCP is often referred to as an additive-increase,multiplicative-decrease(AIMD) algorithm. (Similarly, there are other three terms AIAD, MIAD, MIMD.)

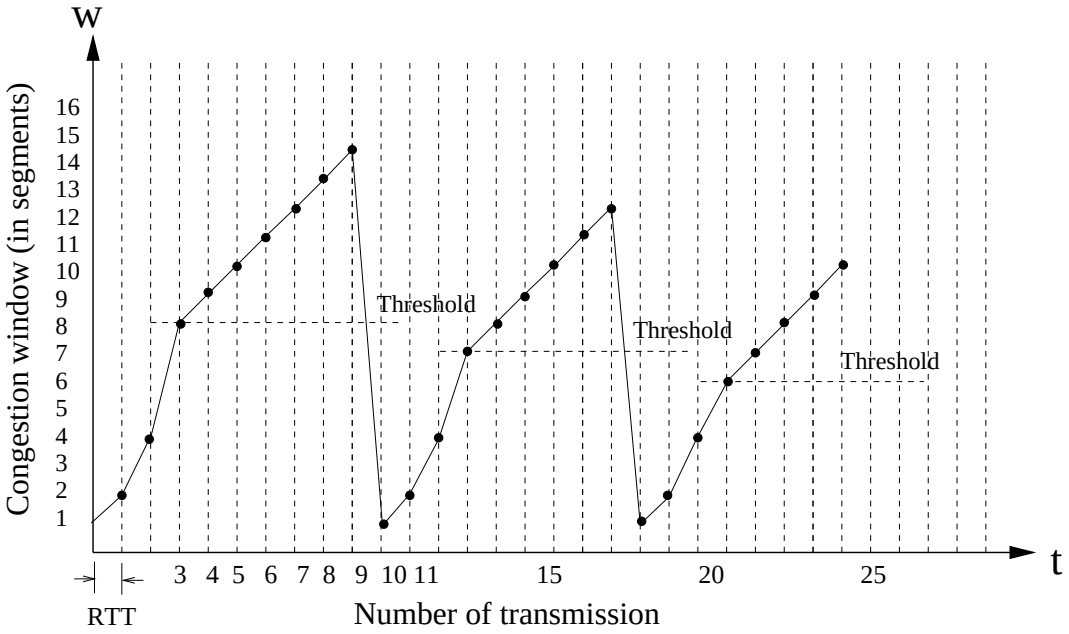

Figure 2: Evolution of TCP Congestion Window

This algorithm process is shown in Figure 2. In this example, the *threshold* is initially set to $8MSS$, The congestion window climbs exponentially fast during slow start and hits the threshold at the third transmission. The congestion window then climbs linearly until loss happens, just after trasmission 9. Note that the current congestion window is $14MSS$, so the new threshold is set to $0.5CongWin = 7MSS$ and the congestion window is set 1. The process goes on.

3 Questions

3.1 Average Throughput

An important measure of the performance of a TCP connection is its throughput, that is the rate at which it transmits data from the snder to the receiver. So next we will discuss in detail what is the average throughput of TCP connection?

For simplicity, we here use a simple model making two assumptions: Ignore "Slow Start" Phase and assume constant $threshold = W$. Under such assumptions, the evolution of congestion window will be as shown in Figure 3.

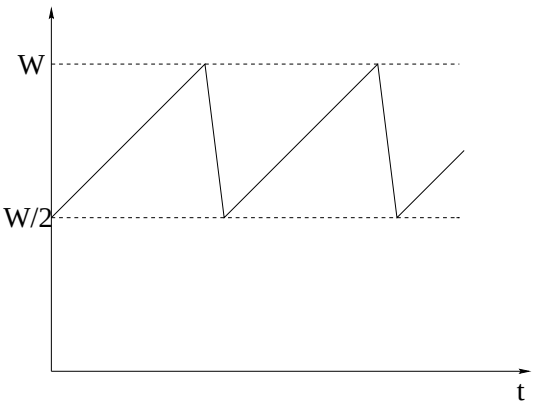

Figure 3: Simple TCP Model

Congestion window starts as $W/2$ and increases linearly by 1 after each transmission until it reaches the $thresholdW$ and starts from $W/2$ again. During this process the congestion window size varies from $W/2$ to W repeatly. There are $W/2+1$ states in all, so the average transmission rate is:

$$\begin{aligned}
\text{Aver. Trans. Rate} &= \frac{1}{W/2+1}\sum_{k=W/2}^{W}\frac{kMSS}{RTT} \\
&= \frac{1}{W/2+1}\Big[\frac{W(W+1)}{2} - \frac{(W/2-1)W/2}{2}\Big]\frac{MSS}{RTT} \\
&= \frac{1}{W/2+1}\frac{0.75W^2+1.5W}{2}\frac{MSS}{RTT} \\
&\approx \frac{1}{W/2}\frac{0.75W^2}{2}\frac{MSS}{RTT} \\
&= \frac{0.75\cdot W\cdot MSS}{RTT}
\end{aligned}$$

We make the approximation because when W is large enough, $W/2 >> 1$ and $0.75W^2 >> 1.5W$, so we can ignore them and get the result.

3.2 Fairness

A goal of TCP congestion-control mechanism is to share a link's bandwidth evenly among multiple TCP connections that are bottlenecked at that link.This is what we referred as Fairness[1]. How TCP additive-increase, multiplicative-decrease algorithm achieves this goal, particularly given that different TCP connections may start different times and thus may have different window sizes at a given point in time?

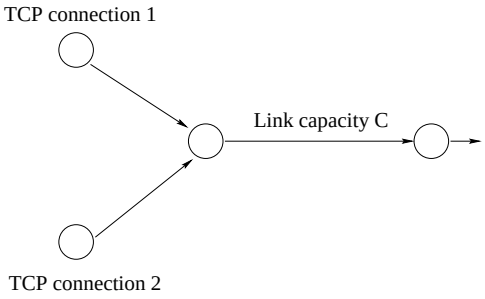

Figure 4: Two TCP connections sharing a single link

Let's consider the simple case as shown in Figure 4. Two connections sharing a single link with transmission capacity C. We assume that the two connections have the same MSS and RTT (So that if they have the same congestion window size, they will have the same throughput), that they have a large amount of data to send, and that no other connections travers this shared link. Also we will ignore the slow-start phase of TCP and assume the TCP connections are operating in additive-increase, multiplicative-decrease algorithm at all times.

Figure 5 plots the throughput realized by the two TCP connections. The 45-degree arrow is the line that the two connections equally share the link bandwidth, Ideally, the sum of the two throughputs should equal C.

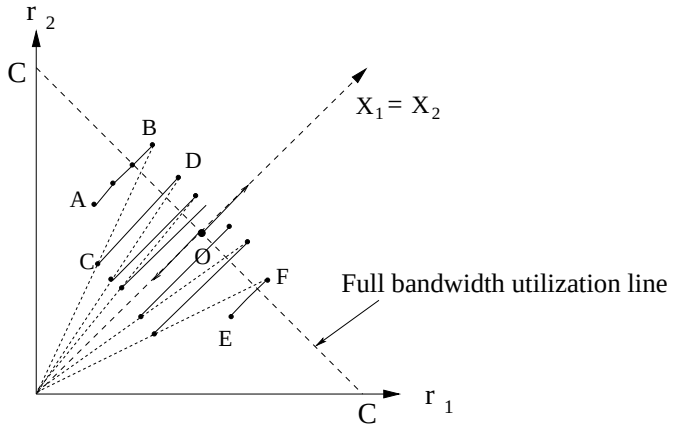

Figure 5: Throughput realized by connections 1 and 2

Suppose that the TCP window sizes are such that at a given point in time, connections 1 and 2 realize throughputs indicated by point A. Because the amount of link bandwidth jointly consumed

[1]For the defination and detail of fairness, see Appendix.

by the two connections is less than C, no loss will happen and both connections will increase their congestion windows by 1 per RTT. Thus, the joint throughput of the two connections proceeds along a 45° line (equal increase for both connections) starting from point A. Eventually, the link bandwidth jointly consumed by the two connections will be greater than C and packet loss will occur. Suppose that connections 1 and 2 experience packet loss when realizing throughput indicated by point B. Connection 1 and 2 then decrease their congestion windows by a factor of two. The resulting throughput realized are thus at point C, halfway along a vector starting at B and ending at the origin. Because the joint bandwidth use is less than C at point C, the two connections again increase their throughputs along a 45° line starting from point C. Eventually, loss will occur again, for example, at point D, and the two connections again decrease their window sizes by a factor two, and so on. Indeally, the bandwidth realized by the two connections eventually fluctuate along the equal bandwidth share line. Similarly, suppose the two TCP window sizes are such that at a given point E, the two congestion window sizes will also vary and eventually fall long the equal bandwidth share line. In fact,the two connections will converge to this behavior no matter which point is as given start point in the two-demensional space.

4 Advantages and Drawbacks

TCP is one of the protocals used in transport layer, compared with other protocals (such as UDP), TCP has its own advantages and disadvantages. next we will discuss in detail.

4.1 Advantages:

The mechanism of TCP congestion control is very simple and easy to impliment. It don't need to make any assumptions about the Network Layer. Moreover it realizes to treat all sessions fairly when it is necessary to turn traffic away from the network. Many network applications run over TCP rather than UDP because they not only want to make use of TCP's reliable transport service but also gets its congestion control to regulate applications' transmission rate. In summary, TCP congestion control has the following advantages:

- Simple and Scalable
- Does not make any assumptions about Network Layer
- Fairness

4.2 Drawbacks:

TCP is not a perfect protocal, it still has some disadvantages need to be improved.

- Based on Packet Loss

  When the sender doesn't receive ACK after timeout, there are usually two main reasons: packet loss or delay. The above algorithm we described is referred as **Tahoe** and there are other two as **Reno** and **Vegas**. The first two algorithms are based on packet loss(Congestion). But when congestion happends, the transmission rate has already been slowed down. We not only want to react to congestion but furthermore also should try to avoid congestion. Vegas, which can improve Reno's preformace, attempts to avoid congestion while maintaining good throughput.

- No QoS-Guarantees

  In TCP congestion control, it only provides a reliable service that guarantee all packets will be arrived in the end no matter how much times it has been transmitted and how long is the delay. So it can not gurantee the service quality such as the transmission rate or the delay etc. Therefore, when a session want to keep its transmission at a certain rate, it can't be realized in TCP. That is why many multimedia applications do not run over TCP for the reason that they do not want their transmission rate throttled, even if the network is very congested.

- Fairness issue
  Sometimes different applications require different fairness, so the fairness realized in TCP may not all the time what we want. For example, there can have different priorities in different applications, such as email can have a low priority because it doesn't matter even if its delay is more than 10 minutes, however, many multimedia applications do not want their transmission

rate throttled, even if the network is very congested. Such as Internet telephone and Internet video conferencing applications prefer to pump their audio and video into the network at a constant rate rather than reduce their rates to "fair" levels at times of congestion.

- Assumes User Cooperation

  TCP congestion control is based on the assumption that users are all cooperated. But it is only an ideal case, we can't guarantee user cooperation. If a user doesn't do as the protocal, but keep or even increase the transmission rate when congestion, then TCP congestion control would fail.

- Vulnerable to UDP sessions

  The multimedia applications running over UDP are not being fair–they do not cooperate with the other connections nor adjust their transmissionrates appropriately. So when TCP sessions and UDP sessions work together, TCP sessions are vulnerable to UDP ones.

- Average Transmission Rate dependent on RTT

  According to the formular we derived before, Aver. Trans. Rate $= \frac{0.75 \cdot W \cdot MSS}{RTT}$, we know that when the window sizes are the same, the transmission rate is in inverse proportional to round-trip time RTT, so TCP "favors" short distance connection because it needs smaller RTT and its corresponding average transmission rate is faster. So from this point of view, it cannot realize the true fairness in the network.

Appendix: Fairness

When multiple sessions share a common link, how to treat all of them fairly when it is necessary to turn traffic away from the network? It is one of the most difficult aspects of flow control. What is the intuition of the defination of fairness? We can define it in such a way that any session is entailed to as much network use as is any other session.

Consider a simple case in a single link. when there are R sessions that share a common link which capacity is C as shown in Figure 6, how to allocate the capacity to each session fairly? We can imagine it as the case how to distribute the cake fairly to R kids during a birthday party. Then you must have the idea that to cut the cake into R equal pieces. It is the same in the transmission link. The fair allocation of the link capacity is to distribute it equally to R sessions and each session has the transmission rate of $X_i = \frac{C}{R}$.

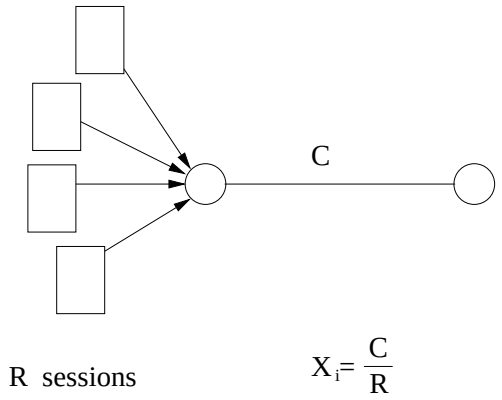

Figure 6: Fairness

The idea of maximizing the network use allocated to the sessions with the minimum allocation leads to the term **Max-Min Fairness**. A way to express the idea is to maximize the allocation of each session i subject to the constraint that an incremental increase in i's allocation does not cause a decrease in some other session's allocation that is already as small as i's or smaller.

Example 1 *Max-Min Fairness*

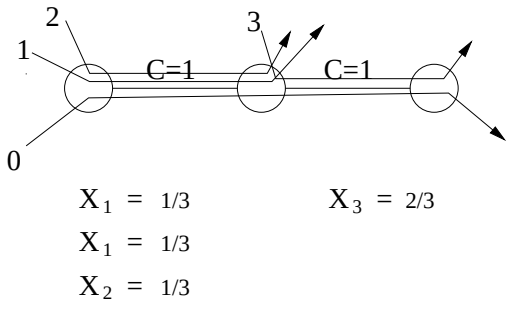

Figure 7: Max-Min Fairness

Figure 7 clarifies some ambiguities in this notion. One session(session 0) flows through the tandem connection of all links and each other session goes through only one link. Suppose the capacity of both links is 1. at the first link, to allocate the network use to session 0,1 and 2 fairly, it should limit each of these three session to a rate of 1/3. The at the second link, based on the rate of session 0, session 3 might be allocated with 2/3 for maxmizing the network use. This is the Max-Min Fairness because now we can't increase any session's allocation because any increase will lead to the decrease of other session's allocation which is as small as it or smaller than it.

Having some idea about the meaning of max-min fairness, now we give out its accurate defination. We assume a directed graph $G = (\mathcal{N}, \mathcal{A})$ for the network and a set of sessions P using the network. Each session p has an associated fixed path in the network. We use p both refer to the session and to its path (if several sessions use the same path, several indices p refer to the same path). Thus, in our model we assume a fixed, single-path routing method.

We denote by r_p the allocated rate for session p. The allocated flow on link a of the network is then

$$F_a = \sum_{\text{all sessions } p \text{ crossing link } a} r_p$$

Letting C_a be the capacity of link a, we have the following constraints on the vector $r = \{r_p \mid p \in P\}$ of allocated rates:

$$\begin{aligned} r_p &\geq 0 \qquad \text{for all } p \in P \\ F_a &\leq C_a \qquad \text{for all } a \in \mathcal{A} \end{aligned}$$

A vector r satisfying these constraints is said to be *feasible*.

A vector of rates r is said to be *max-min fair* if it is feasible and for each $p \in P$, r_p cannot be increased while maintaining feasible without decreasing $r_{p'}$ for some session p' for which $r_{p'} \leq r_p$. (More formally, r is max-min fair if it is feasible, and for each $p \in P$ and feasible $\bar{r}$ for which $r_p < \bar{r_p}$, there is some $p_\prime$ with $r_p \geq r_{p'}$ and $r_{p'} > \bar{r_{p'}}$.) Our problem is to find a rate vector that is max-min fair.

Given a feasible rate vector r, we say that link a is a *bottleneck link* with respect to r for a session p crossing a if $F_a = C_a$ and $r_p > r_{p'}$ for all sessions p' crossing link a. Figure 8 provides an example of a max-min fair rate vector and illustrates the concept of a bottleneck link.

Example 2 *Max-Min Fair and bottleneck link*

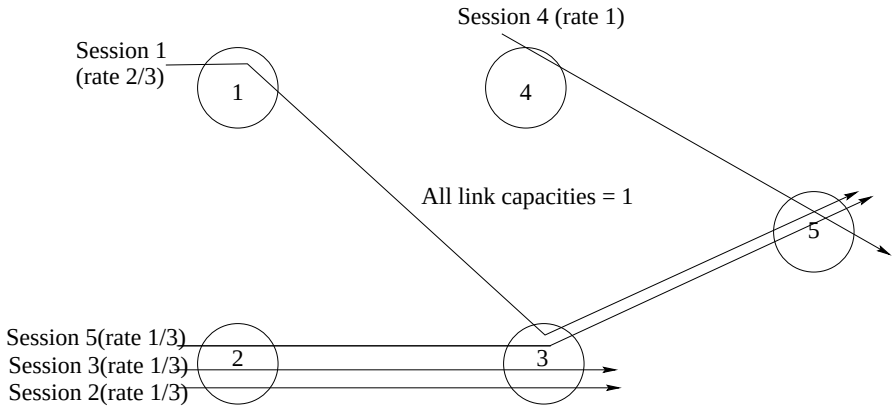

Figure 8: Max-min fair and bottleneck link

This figure gives a max-min solution to an example network. The bottleneck links of sessions 1,2,3,4 and 5 are (3,5),(2,3),(2,3),(4,5),and (2,3), respectively. Link (3,5) is not a bottleneck link for session 5 since sessions 1 and 5 share this link and session 1 has alarger rate than session 5. Link (1,3) is not a bottleneck link of any session since it has an excess capacity of 1/3 in the fair solution.

In the above example, every session has a bottleneck link. It turns out that this property holds in general as shown in the following proposition:

Proposition. A feasible rate vector r is max-min fair if and only if each session has a bottleneck link with respect to r.