Chapter 4

TCP Switching

4.1 Introduction

In Chapters 1, 2 and 3, I have argued that the Internet would benefit from having more circuit switching in the core of a network that uses packet switching everywhere else. With such an architecture, carriers obtain an infrastructure that is reliable and cost-effective, that can scale to meet the growth demands of Internet traffic, that provides quality of service guarantees, and that does not deteriorate the response time end users currently receive from the network.

Now we need to check that it is not too burdensome to implement a circuit-switched solution for the core. The main concerns arise from the characteristics that set circuit switching apart from packet switching. Namely, won't the amount of state be too large or complex to be handled in real time? Isn't the bandwidth and processing overhead of circuit management too large? What is the effect of the bandwidth inefficiencies and call blocking probabilities of circuit switching? This chapter will look at these and other issues.

As pointed out in Chapter 1, circuit switches are already used in the core of the network in the form of SONET, SDH and DWDM switches. However, IP treats these circuits as static, point-to-point links connecting adjacent nodes; the physical circuits and IP belong to different layers, and they are completely decoupled since they operate autonomously and without cooperation. Decoupling of layers has many

advantages. It lets the circuit-switched physical layer evolve independently of IP and vice versa. IP runs over a large variety of physical layers regardless of the underlying technology. At the same time, much of repetition exists between the packet-switched IP layer and the circuit-switched physical layer. For example, a network must route both IP datagrams and circuit paths, yet they use different routing protocols, and their implementations are incompatible. This makes simple and obvious operations infeasible. As a result, provisioning of circuits is done manually, and it can take weeks to allocate a new circuit or to change the capacity of an existing one. Circuit allocation is also rather inflexible because the circuit capacity has to be a multiple of a coarse STS-1 channel (51 Mbit/s). Consequently, circuit provisioning is inefficient and slow to react in real time to changes in traffic patterns.

This chapter and the next present two different ways of integrating circuit switching and packet switching in an evolutionary fashion; that is, these chapters show how end hosts and edge routers are not required to change their protocol stacks or add new signaling mechanisms. This chapter focuses on the mapping of application flows to fine-grain circuits using lightweight signaling, whereas the next chapter maps inter-router flows to coarse circuits with heavyweight signaling.

Below, I present one of the main contributions of this thesis: I propose a technique, called TCP Switching, that exposes circuits to IP; each application flow triggers its own end-to-end circuit creation across a circuit-switched core. TCP Switching takes its name from, and strongly resembles, IP Switching [129], in which a new ATM virtual circuit[1] is established for each application-level flow. Instead, TCP Switching maps these flows to true circuits, thus reaping the advantages of circuit switching.

The proposed architecture is called TCP Switching because most flows today (over 90%) are TCP, and so this architecture is optimized for the common case of TCP connections; but this technique is not limited to TCP, and any uni- or bidirectional flow can be accommodated, albeit less efficiently.

TCP Switching requires no additional signaling, as the first observed packet in a flow triggers the creation of a new circuit. It incorporates modified circuit switches that use existing IP routing protocols to establish circuits. Routing, thus, occurs hop

[1]It is worth noting that virtual circuits are just a connection-oriented packet-switching technique.

by hop, and circuit maintenance uses soft state; that is, it is removed through an inactivity timeout.

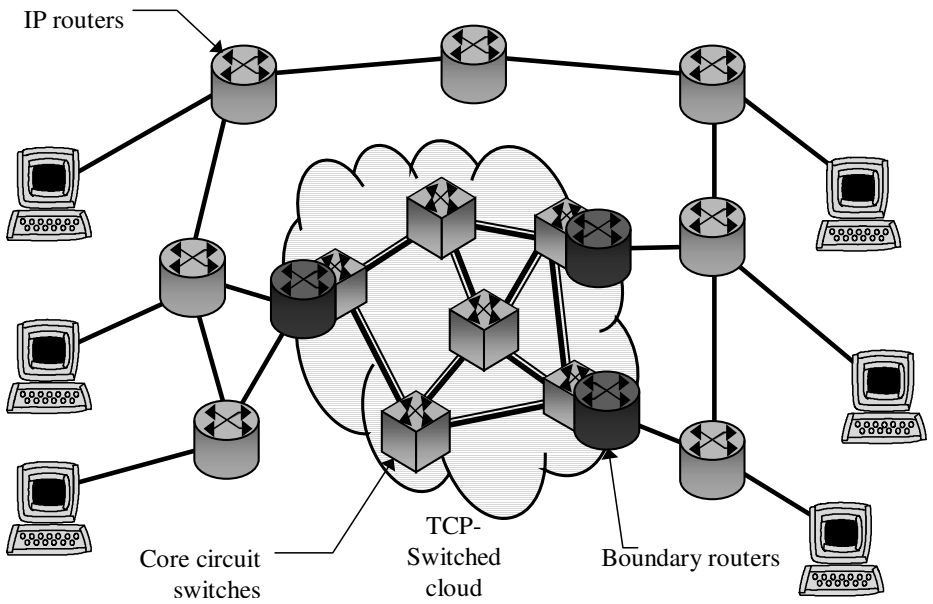

Figure 4.1: An example of a TCP-Switching network.

TCP Switching can be deployed incrementally in the current Internet by creating self-contained TCP-Switching clouds inside a packet-switched network, as Figure 4.1 shows. The packet-switched portion of the network remains unchanged. The core of the circuit-switched portion of the network is built from pure circuit switches (such as SONET cross connects) with simplified signaling to create and destroy circuits. Boundary routers act as gateways between the domains of packets and circuits, and they are most likely conventional routers with circuit-switched line cards.

We are interested in how to make circuit switches and IP routers cooperate. TCP Switching presents a method of interaction, enabling automatic and dynamic circuit allocation. Needless to say, TCP Switching is not the only way of integrating circuit switching and packet switching in the Internet. Indeed, there are several other approaches that I will describe in Chapter 6.

4.1.1 Organization of the chapter

Section 4.2 summarizes the advantages of circuit switching and then describes the potential disadvantages and pitfalls of circuit switching. Section 4.3 describes the network architecture of TCP Switching. Next, Section 4.3.1 analyzes what a typical application flow is in the Internet. Based on these observations about flows and on the discussion in Section 4.2, Section 4.3.3 makes some design choices for TCP Switching. Section 4.3.4 describes the results of the implementation of a TCP Switch. Section 4.4 provides some discussion of the proposed architecture. Finally, Section 4.5 concludes this chapter.

4.2 Advantages and pitfalls of circuit switching

Let us review the main advantages of circuit switching that were described in Chapters 1, 2 and 3:

- Lack of buffers in the data path (Chapter 1).
- Possibility of all-optical data paths (Chapter 1).
- Higher switching capacity (Chapter 2).
- Simple and intuitive QoS (Chapter 2).
- Simple admission control (Chapter 2).
- No degradation of the response time (Chapter 3).

4.2.1 Pitfalls of circuit switching

Despite the advantages listed above, circuit switching has some potential implementation problems that may preclude its utilization if they prove to be too cumbersome. However, I will argue in this chapter that with the proper implementation they are not significant enough to prevent the adoption of circuit switching in the core of the Internet.

4.2.2 State maintenance

Circuit switching requires circuits and their associated state to be established before data can be transferred. A large number of circuits might require a circuit switch to maintain a lot of state. In practice, by observing real packet traces (see Section 4.3.1), I have found that the number of flows, and the rate at which they are added and removed, to be quite manageable in simple hardware using soft state. This holds true even for a high-capacity switch.

4.2.3 Signaling overhead and latency

In order to set up and tear down circuits, switches need to exchange information in the form of signaling. This signaling may represent an important overhead in terms of bandwidth or processing requirements. Depending on how inactive circuits are removed, this state is said to be hard or soft state. If it is hard state, then maintenance is complex because it requires explicit establishments and teardowns, and it has to take into account Byzantine failure modes. In contrast, soft state is simpler to maintain because it relies on end hosts periodically restating the circuits that they use. If a circuit remains idle for a certain period of time, it is timed out and deleted. With the use of hard or soft state, there is a tradeoff between signaling complexity and signaling overhead.

In addition, a considerable latency may be added if additional handshakes are required to establish a new circuit. As I will show with TCP Switching, it is possible to avoid any signaling overhead or latency with circuit switching by piggybacking on the end-to-end signaling that already exists in most user connections.

4.2.4 Wasted capacity

Circuit switching requires circuits to be multiples of a common minimum circuit size. For example, SONET commonly cross connects to provision circuits in multiples of STS-1 (51 Mbit/s). Having flows whose peak bandwidth is not an exact multiple wastes link capacity. Yet using smaller circuit granularity increases the amount of

state maintained by the switch. In addition, because bandwidth is reserved, capacity is wasted whenever the source idles and the circuit is active.

In any case, network carriers do not seem to worry much about bandwidth inefficiencies, since networks today are lightly used, and they will likely remain that way since carriers are more interested in operating a reliable network than an efficient one, as shown in Chapter 2. Furthermore, the wasted capacity is not a problem if the speedup of circuit switches with respect to packet switches is bigger than the bandwidth inefficiency.

4.2.5 Blocking under congestion

If no available circuit exists in circuit switching, any new circuit request cannot be processed (gets blocked) until a circuit is free. This data flow system works differently from the link-sharing paradigm present in the Internet today, in which packets will still make (albeit slow) progress over a congested link. However, as we saw in Chapter 3, this blocking does not affect the end-user response time. In the circuit-switched core, some flows may take a longer time to start, but, on average, they finish at the same time as the packet-switched flows.

4.3 TCP Switching

TCP Switching consists of establishing fast, lightweight circuits triggered by application-level flows. Figure 4.1 shows a self-contained TCP Switching cloud inside the packet-switched Internet. The network's packet-switched portion does not change, and circuit switches, such as SONET crossconnects, make up the core of the circuit-switching cloud. These circuit switches have simplified mechanisms to set up and tear down circuits. Boundary routers are conventional routers with circuit-switched line cards, which act as gateways between the packet switching and circuit switching.

The first arriving packet from an application flow triggers the boundary router to create a new circuit. An inactivity timeout removes this circuit later. Hence, TCP Switching maintains circuits using soft state.

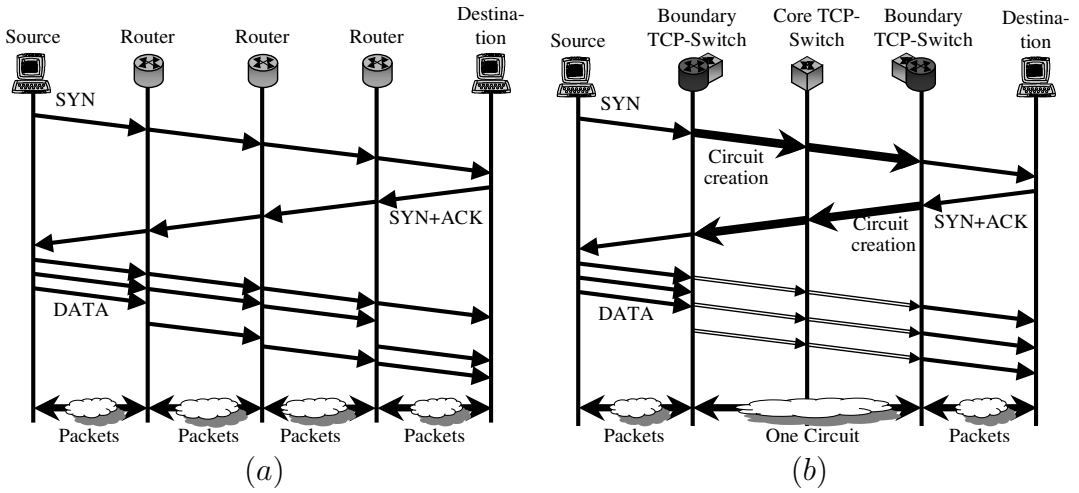

Figure 4.2: Sample time diagram of (a) a regular TCP connection over a packet-switched Internet, and (b) a TCP connection traversing a TCP Switching cloud. The network topology is shown in Figure 4.1.

In the most common case, the application flow is a TCP connection, where a SYN/SYN-ACK handshake precedes any data exchange, as shown in Figure 4.2a. In this case, the first packet arriving at the boundary router is a TCP synchronization (SYN) packet. This automatically establishes an unidirectional circuit as part of the TCP connection setup handshake, and thus no additional end-to-end signaling mechanism is needed, as shown in Figure 4.2b. The circuit in the other direction is established similarly by using the SYN-ACK message. By triggering the circuit establishment when the router detects the first packet — whether or not it is a TCP SYN packet—, TCP Switching is also suitable for non-TCP flows and for on-going TCP flows that experience a route change in the packet-switched network. This is why TCP Switching, despite its name, also works for the less common case of UDP and ICMP user flows.

An examination of each step in TCP Switching, following Figure 4.2b, shows how this type of network architecture establishes a circuit end to end for a new application flow. When the boundary router (shown in Figure 4.3) detects an application flow's first packet, it examines the IP packet header and makes the usual next-hop routing decision to determine the outgoing circuit-switched link. The boundary router then

checks for a free circuit on the outgoing link (for example, an empty time slot or an unused wavelength). If one exists, the boundary router begins to use it, and forwards the packet to the first circuit switch in the TCP Switching cloud. If no free circuits exist, the protocol can buffer the packet with the expectation that a circuit will become free soon, it could evict another flow, or it could just drop the packet, forcing the application to retry later. Current implementations of TCP will resend a SYN packet after several seconds, and will keep trying for up to three minutes (depending on the implementation) [19].

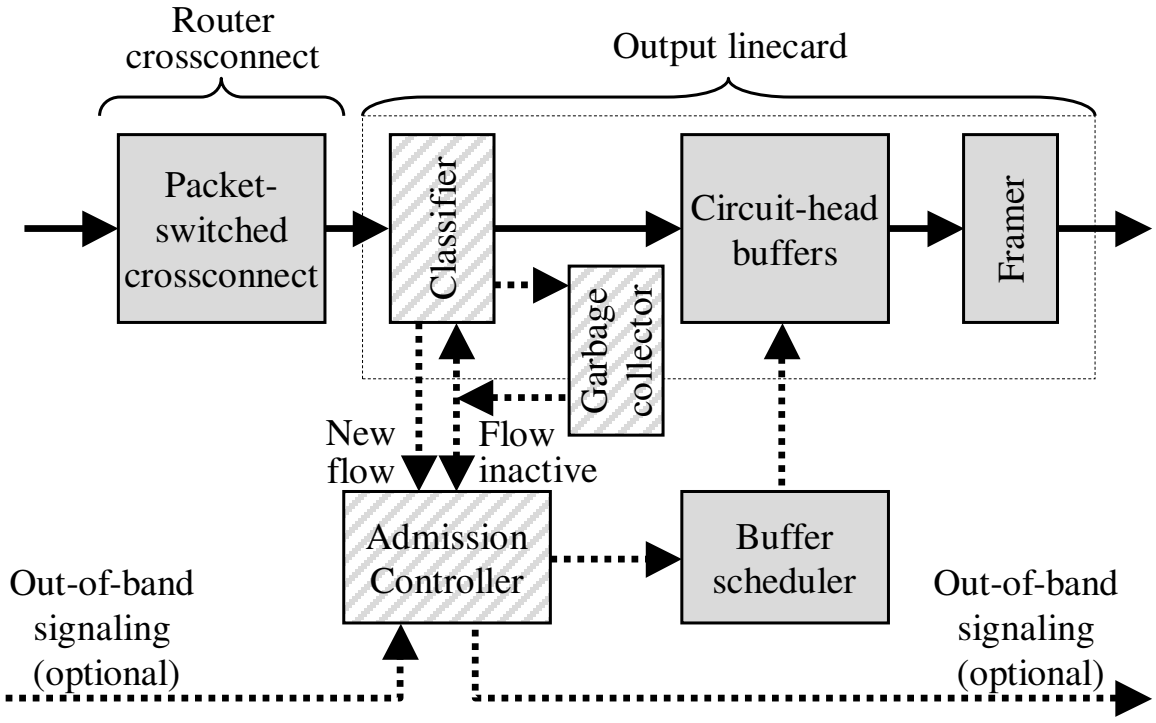

Figure 4.3: Functional block of a TCP-Switching boundary router. The data path is represented by continuous arrows, the control path by the dashed ones. The shaded blocks are not present in a regular router. The classifier and the garbage collector are shown in the output linecard, but they could also be part of the input linecard.

If the circuit is successfully established on the outgoing link, the packet is forwarded to the next-hop circuit switch. The core circuit switch (shown in Figure 4.4) will detect that a previously idle circuit is in use. It then examines the first packet on the circuit to make a next-hop routing decision using its IP routing tables. If a free

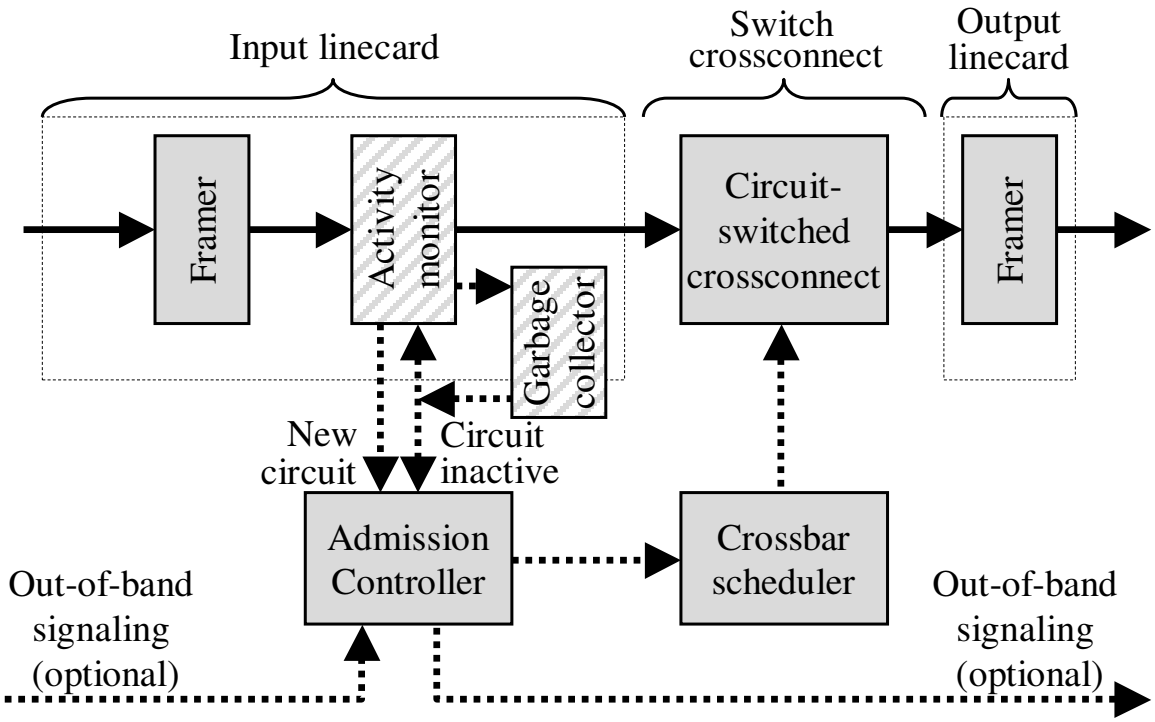

Figure 4.4: Functional block of a TCP-Switching core circuit switch. The data path is represented by continuous arrows, the control path by the dashed ones. The shaded blocks are not present in a regular circuit switch.

outgoing circuit exists, it connects the incoming circuit to the outgoing circuit. From then on, the circuit switch does not need to process any more packets belonging to the flow.

The circuit establishment process continues hop by hop across the TCP-Switching cloud until, hopefully, the circuit is established all the way from the ingress to the egress boundary router. The egress boundary router receives packets from the circuit as they arrive, determines their next hop, and sends them across the packet-switched network toward their destination.

In its simplest form, TCP Switching allows all boundary routers and circuit switches to operate autonomously. They can create circuits, and remove (timeout) circuits, independently. Obvious alternative approaches include buffering the first packet while sending explicit signals across the circuit-switched cloud to create the circuit. However, this removes autonomy and complicates state management, and so

it is preferable to avoid this method.

The boundary-router and the circuit-switch complexities are minimal, as shown in Figures 4.3 and 4.4. The ingress boundary router performs most packet processing. It has to map incoming packets from existing flows into the corresponding outgoing circuits, like any flow-aware router would. Additionally, the ingress boundary router processes new flows: it must recognize the first packet in a flow and then determine if the outgoing link has sufficient capacity to carry the new circuit; in other words, it has to do admission control. On the other hand, core circuit switches only need to do processing once per flow, rather than once per packet. These circuit switches only require a simple activity monitor to detect new (active) circuits and expired (idle) circuits. Alternatively, a design could use explicit out-of-band signaling in which the first packet is sent over a separate circuit (or even a separate network) to the signaling software on the circuit switch. In this case, hardware changes to the circuit switch are not necessary because the activity monitor and the garbage collector would not be needed.

Recognizing the first packet in a new flow requires the boundary router to use a four-field, exact-match classifier using the (*source IP address, destination IP address, source port, destination port*) tuple. This fixed-size classifier is very similar to the one used in gigabit Ethernet, and it is much simpler than the variable-size matching that is used in the IP route lookup. When packets arrive for existing circuits, the ingress boundary router must determine which flow and circuit the packet belongs to (using the classifier). The short life of most flows requires fast circuit establishment and tear down.

4.3.1 Typical Internet flows

To provide an understanding of the feasibility and sensibility of TCP Switching, I now study some of the current characteristics of Internet traffic in the backbone. This section focuses on application flows, since TCP Switching establishes a circuit for each application flow. More precisely, I start discussing what a TCP flow is, and how it behaves, because over 90% of Internet traffic is TCP — both in terms of

packets, bytes and flows. I have studied `traceroute` measurements, as well as packet traces from OC-3c and OC-12c links in the vBNS backbone network[2] [131]. These results are similar to the ones obtained from flow traces from OC-48c links in the Sprint backbone [170].

Traffic characteristics	80-percentile	Average	Median
TCP flow duration (seconds)	$\leq 4-10$	$\leq 3-7$	$\leq 0.5-1.2$
Packets per flow	≤ 12	$\leq 10-200$*	$\leq 5-9$
Flow size (Kbytes)	$\leq 2.5-4$	$\leq 9-90$*	$\leq 0.6-1.3$
Flow average bandwidth [size/duration] (Kbit/s)	$\leq 50-100$	$\leq 20-140$*	$\leq 8-15$
Fraction of flows with retransmissions	$\leq 7.8\%$	$\leq 5.6\%$	$\leq 4.7\%$
Fraction of flows experiencing reroutes	$\leq 0.19\%$	$\leq 0.39\%$*	$\leq 0.02\%$
Asymmetrical connections	Around 40% of the flows transmit an ACK after the FIN; i.e., they keep acknowledging data packets that are sent in the other direction.		

Table 4.1: Typical TCP flows in the Internet. The figures indicate the range for the 80-percentile, the average and the median for the different links taken in August, 27-31, 2001 [131]. The magnitudes marked with * present a non-negligible amount of samples with very high values; that is, their statistical distribution has long and heavy tails. This is why the average is higher than the 80-percentile.

Table 4.1 describes the typical TCP flow in the Internet. TCP connections usually last less than 10 seconds, carry less than 4 Kbytes of data and consist of fewer than 12 packets in each direction. Less than 0.4% of connections experience a route change. The typical user requests a sequence of files for downloading and wants the fastest possible download for each file. In most cases, the requested data is not used until the file has completely arrived at the user's machine.

Figure 4.5 shows the cumulative histogram of the average flow bandwidth —

[2]This data was made available through the National Science Foundation Cooperative Agreement No. ANI-9807479, and the National Laboratory of Applied Network Research.

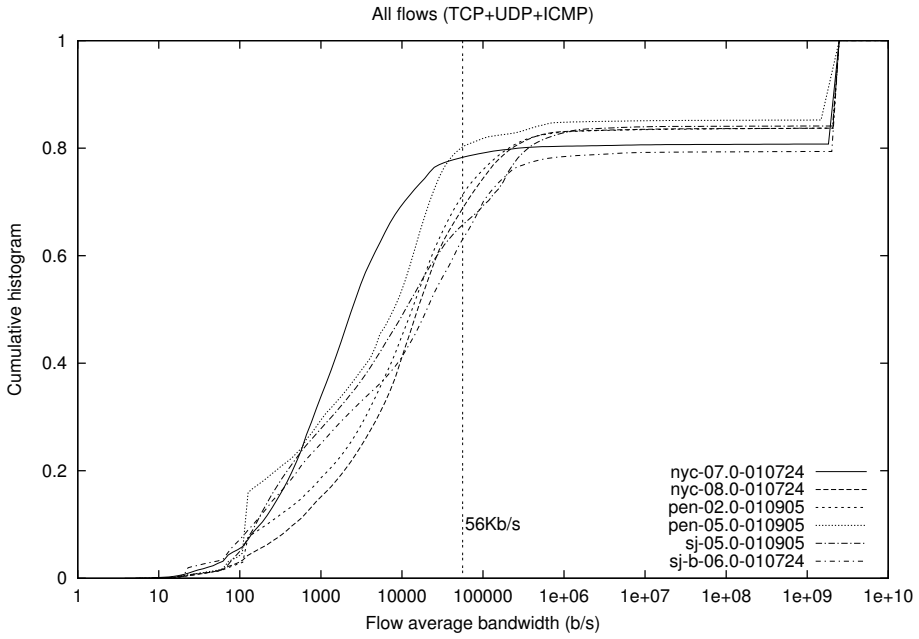

Figure 4.5: Cumulative histogram of the average flow bandwidth for TCP and non-TCP flows. The traces were taken in July and September 2001 from OC-48c links in the Sprint backbone network [170]. The flows with the peak bandwidth (2.5 Gbit/s) are single-packet flows (usually UDP and ICMP flows, and a few broken TCP connections).

defined as the ratio between the flow size[3] and the flow duration — for both TCP and non-TCP flows from several OC-48c traces from the core of the Internet. As one can see, with the exception of single-packet flows, very few flows achieve an average bandwidth that is greater than 1 Mbit/s. Furthermore, most of the multi-packet flows (78%-97% of them) receive less than 56 Kbit/s from the network either because one of the access links is a 56-Kbit/s modem or because the application does not take advantage of all the available bandwidth. This confirms that, as discussed in Chapter 3, the bandwidth ratio between core and access links, N, is much greater than 500.

Figure 4.6 shows the correlation between the flow duration and the flow size. Most flows are both short in size (80-4000 bytes) and medium in duration (0.1-12 s). This

[3]The flow size is the number of bytes transported by the flow, including the header overhead, control messages and retransmissions.

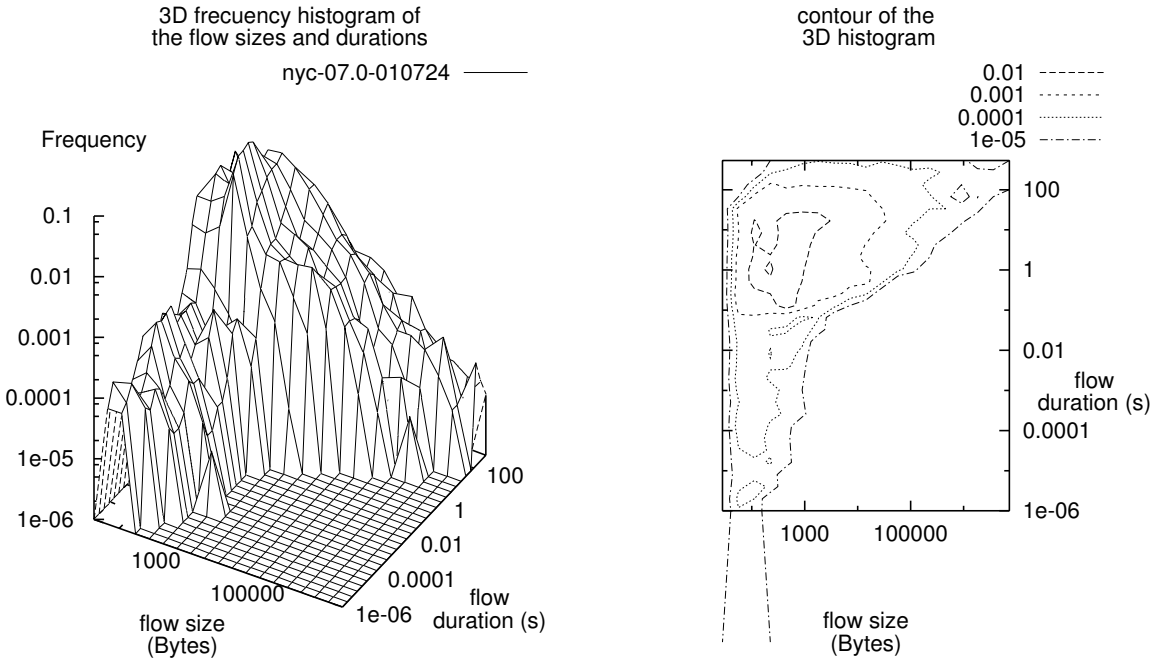

Figure 4.6: 3D Frequency histogram of flow sizes and durations for both TCP and non-TCP flows in one trace from OC-48c links in the Sprint backbone network [170].

is because the source cannot fill up the core link on its own; the slow-start phase of the TCP connections requires several round trips before the source can transmit at the available rate for the flow, and, in addition, the access link forces spacing between consecutive packets belonging to the same flow.

4.3.2 Design options

TCP Switching is in fact a family of network architectures in which there are numerous design options. These options indicate tradeoffs between implementation simplicity, traffic control and efficiency. Below, I list several of these design options:

Circuit establishment

Option 1	Triggered by first packet seen in a flow (can be any packet type).
Option 2	Triggered by TCP SYN packets only.
Notes	If there is a path reroute outside the TCP switched cloud, the switch will not detect the SYN packet. This is rare in practice.

Circuit release

Option 1	Triggered by inactivity timeout (soft state).
Option 2	Triggered by a finish (TCP FIN) signal (hard state).
Notes	Neither option is perfect. The switch might sever connections that either have asymmetrical closings (hard state) or long idle periods (soft state).

Handling of non-TCP flows

Option 1	Treats user datagram protocol (UDP) and TCP flows the same way.
Option 2	Multiplex UDP traffic into permanent circuits between boundary routers.
Notes	UDP represents a small (but important) amount of traffic.

Signaling

Option 1	None. Circuit establishment is implicit based on observed packets.
Option 2	Explicit in-band or out-of-band signaling to establish and remove circuits.
Notes	In-band signaling requires no additional exchanges, but it is more complex to implement.

Circuit routing

Option 1	Hop-by-hop routing.
Option 2	Centralized or source routing.
Notes	A centralized algorithm can provide global optimization and path diversity, but it is slower and more complex.

Circuit granularities

Option 1	Flat. All switches have the same granularity.
Option 2	Hierarchical. Fine circuits are bundled in coarser circuits as we move towards the inner core.
Notes	A coarser granularity means that the switch can go faster because it has to process less.

4.3.3 Design choices

Using the observations in Section 4.3.1, I now describe some design choices that I have used in experiments of TCP Switching.

Circuit signaling

In my design, I use implicit signaling, that is, the arrival of a packet on a previously inactive circuit triggers the switch to route the packet and create a new circuit. Circuits are removed after they have been idle for a certain period of time. This eliminates any explicit signaling at the small cost of adding a simple activity monitor to the data path.

Bandwidth assignment

I assume in the experiments that the core circuit switches carry 56-Kbps circuits to match the access links of most network users. High capacity flows use multiple circuits. There are two ways of assigning a peak bandwidth to a flow: the preferred one is to make the decision locally at the ingress boundary router. The alternative is to let the source use an explicit signaling mechanism like RSVP [18] or some TCP-header option, but this requires a change in the way applications currently use the network. With the local bandwidth assignment, users would be allocated 56-Kbit/s by default unless their address appears in a local database listing users with higher data-rate access links and/or who have paid for a premium service.

Flow detection

The exact-match classifier detects new flows at the ingress boundary router. The classifier compares the headers of arriving packets against a table of active flows to check if the flow belongs to an existing circuit, or whether a new circuit needs to be created. The size of the classifier depends on the number of circuits on the outgoing link. For example, an OC-192c link carrying 56-Kbps circuits requires 178,000 entries in its table, an amount of state that fits on an on-chip SRAM memory. Given the duration of measured flows, in a one-second period one expects about 31 million lookups, 36,000 new connections, and 36,000 old connections to be timed out for an OC-192c link. This is quite manageable in dedicated hardware [4, 71].

I use soft state and an inactivity timer to remove connections. For TCP flows, an alternative could be to remove circuits when the router detects a FIN signal, but in about 40% of TCP flows, acknowledgement (ACK) packets arrive after the FIN because the communication in the other direction is still active.

Inactivity timeouts

In my design, the timeout duration is a tradeoff between efficiency in bandwidth and signaling. For example, my simulations suggest that a 60-second timeout value will reliably detect flows that have ended (which is similar to results by IP Switching [129] and Feldman et al. [74]). This timeout value ensures that flows are neither blocked nor severed during the connection lifetime.[4] But, the cost of using such a long timeout value is high because the circuit remains unused for long time, especially if the flow duration is of only a few seconds.

To reduce the bandwidth inefficiencies, one could use a very short timeout value so that there is some statistical multiplexing among active flows. However, if the timeout is very short, the control plane of circuit switching would often have to be visited more than two times during the lifetime of a flow. In the extreme case of a timeout of zero, circuit switching degenerates into packet switching, where every

[4]If the circuit were timed out during the lifetime of the flow, it can be reestablished rapidly. However, there is a risk that a new request gets rejected because the old resources have been claimed by another flow.

piece of information has to be routed, processed and buffered, which severely limits the switch performance. If one wants to avoid this degenerate behavior, the timeout should be greater than the maximum transmission time of a packet through the circuit (214 ms for 56-Kbit/s circuits, 8 ms for 1.5-Mbit/s circuits).

To choose the right timeout value, one has to take into account the timing of the TCP mechanisms to avoid severing the circuit during a naturally occurring pause. The first observation would be that the minimum retransmission timeout value of TCP is 1 s [145]. However, retransmission timeouts are rare, they represent less than 0.5% of all transmissions, and so it should not be very expensive to have inactivity timeouts of less than 1 s.

A more important factor is the slow-start mechanism used by all TCP connections to ramp-up the flow rate. This mechanism creates some silence periods that occur during the initial round trips. Having an inactivity timeout that is smaller than the round trip time (RTT) is very expensive, especially for the frequent short TCP flows (the so-called *mice* [23]). It is then recommended that inactivity timeout values greater than the RTT be used (on earth, most RTTs for minimum-size packets are smaller than 250 ms).

Circuit replacement policies

In my experiments, when a circuit remained inactive for a certain period of time it was torn down. Circuits that time out need not be evicted immediately; they may just be marked as candidates to be replaced by a new circuit when a request arrives. This reduces the per-circuit processing for circuits that are incorrectly marked as inactive. If the new circuit request uses the same path, it is then possible to reuse the existing circuit without any new signaling. One could use different replacement policies, as with the cache of a computer system. The simplest policy is the Least Recently Used (LRU), but others are possible. In case of contention, preemptive policies could be used to evict lower-priority circuits to accommodate higher-priority ones.

Switching unit

Several applications, such as web browsing, open several parallel TCP connections between the same two end hosts. These parallel flows share the same access link, and thus it would be wasteful to allocate the bandwidth of the access link to each of them. Instead, all these parallel flows should be sharing a single circuit. So, rather than using TCP flows as the switching unit, it is better to use IP flows (i.e., flows between pairs of end hosts).

4.3.4 Experimentation with TCP-Switching networks and nodes

I experimented with TCP-Switching networks via simulation using ns-2 [89]. The main results are presented in Section 3.5, and they show that TCP Switching does not yield a worse response time than packet switching for the core of the network, despite the bandwidth inefficiencies and call blocking that are typical of circuit switching.

These simulations assume that TCP Switching nodes can process the requests for new circuits as quickly as needed. This hypothesis was validated through the implementation of a TCP Switching boundary router.[5] The boundary router was implemented as a kernel module in Linux 2.4 running on a 1-GHz Pentium III. Neither this platform nor the implementation were particularly optimized to perform this task, and yet in TCP Switching forwarding a packet in the boundary router took $17-25\ \mu s$ (as opposed to 17 μs for regular IP forwarding and the $77-115\ \mu s$ of IP's QoS forwarding that comes standard with Linux). In this non-optimized software, the circuit setup time is approximately 57 μs, fast enough to handle new connection requests of an OC-48c link (an OC-192c link has an average flow interarrival time of $16-39\ \mu s$ at full capacity, an OC-48c of $64-156\ \mu s$). These numbers should drop dramatically if part of the software were implemented in dedicated hardware.

[5]This prototype was built by Byung-Gon Chun, an M.S. student at Stanford, for a 10-week class project under my supervision [39].

4.4 Discussion

TCP Switching exploits the fact that most of our communications are connection oriented and reliable. Rather than using complex signaling mechanisms to create and destroy circuits, TCP Switching piggybacks on existing end-to-end mechanisms to manage circuits. More specifically, TCP Switching uses the initial handshake of the most common type of flows, TCP connections, to create a circuit in the core of the network. When a circuit request message gets dropped, TCP Switching relies on the TCP retransmission mechanisms to set up the circuit again at some later time.

In addition, TCP Switching tries to exploit some of the statistical multiplexing that exists among flows. Obviously, it does not achieve the statistical multiplexing of packet switching because, if a flow does not fully utilize its circuit capacity, this unused bandwidth is wasted. However, TCP Switching does not reserve resources for inactive flows, and so those resources can be employed by other active flows. In this way, TCP Switching achieves some statistical multiplexing gain.

TCP Switching is indeed an extreme technology. In what follows, I will discuss some of the concerns that arise when this approach is described; namely, I will discuss the impact of single-packet flows, bandwidth inefficiencies and denial of service.

4.4.1 Single-packet flows

The longer flows are, the more efficient TCP Switching because the circuit setup cost is amortized over the longer data transfer. It is unclear how flow sizes will evolve in the future. On one hand, there is a trend for longer flows from downloads and streaming of songs and video. On the other hand, traffic from sensors is likely to consist of very short exchanges, perhaps consisting of single-packet flows. Even though most probably those single-packet flows will be aggregated before being sent through the Internet backbone [2], it is worth asking what would happen if single-packet flows became a large fraction of Internet traffic.

As mentioned in Section 4.3.3, packet switching can be considered to be a special case of circuit switching in which all flows are 1-packet long. The processing and forwarding of a circuit request in the control plane of a circuit switch is similar to the

forwarding of a packet in the data plane of a router. When a circuit arrives, a next-hop lookup has to be performed and resources have to be checked. If these resources are available, the crossconnect needs to be scheduled; otherwise, the request has to be buffered or dropped. The only difference with packet switching is that state is maintained so that next time data arrives for that circuit, the data path can forward the information without consulting the control plane.

In TCP Switching, single-packet flows are forwarded as if using packet switching by the control plane, while long flows are forwarded by the data path of circuit switching at a much higher rate. In order to avoid interactions in the control plane between these two classes of flows, one can create two separate queues and process them differently (e.g., the single-packet flows would not write any state).

4.4.2 Bandwidth inefficiencies

In TCP Switching, the wastage of bandwidth is evident because it suffers from acute fragmentation; the bandwidth allocated to a circuit is reserved for its associated flow, and if the flow does not use it, the bandwidth is wasted. Nevertheless, it is not clear to what extent the bandwidth inefficiency is a problem because, as shown in Figure 1.3, optical-link capacity does not have the technology limitations of buffering and electronic processing. One should then ask the question, how much speedup is needed in a circuit switch to compensate for the wasted bandwidth?

In order to quantify how much bandwidth remains unused, we need to look at the time diagram of a typical circuit, as shown in Figure 4.7. As we can see, during the lifetime of a TCP-Switching circuit, there are three phases when bandwidth is wasted by a TCP flow: (1) during the slow start phase, when the source has not yet found the available bandwidth in the circuit, (2) during the congestion avoidance phase, and (3) during the inactivity period that is used to timeout and destroy the circuit. The total amount of bandwidth that is wasted in each phase will depend on the source activity, the flow length, the round-trip time, and the inactivity timeout. For example, the so-called TCP "mice" or "dragonflies" [23] are so short that they do not enter the congestion avoidance phase.

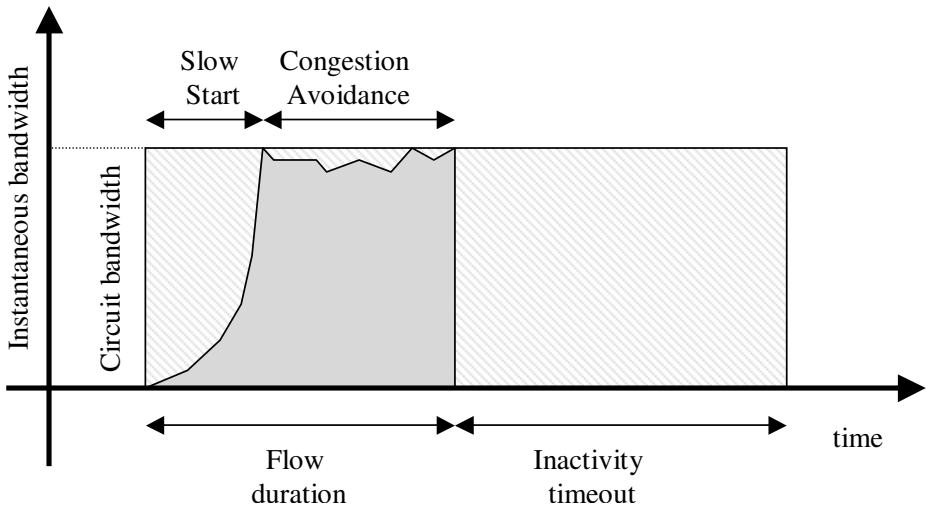

Figure 4.7: Bandwidth inefficiencies in TCP Switching. The dashed circuit bandwidth is wasted.

Given that application flow sizes are typically shorter than 10 s, an inactivity timeout of 60 s such as the one proposed in [129, 74] is extremely wasteful. Better efficiencies can be achieved with a timeout value that is a little larger than the RTT (as proposed in Section 4.3.3) because this timeout value is comparable to the duration of the slow-start phase, which lasts a few RTTs.

In any case, traffic is highly asymmetric. Usually, one end host holds the information, and the other simply downloads it, such as with web browsing. This means that one direction of the connection will be filling up the pipe with large packets (typically of 1500 bytes), whereas the other direction will be sending 40-Byte long acknowledgements. If the two circuits belonging to a bi-directional flow are symmetric, then even if we achieved a bandwidth efficiency of close to 100% in the direction of the download, the reverse direction will get an efficiency of less than 2.7%. The overall efficiency would be only 51%. However, the direction of download is not uniformly distributed, as servers tend to be placed in PoPs and co-location facilities, and so the direction in which the bottleneck occurs will get a bandwidth efficiency closer to 100% than to 2.7%.[6] If the bandwidth inefficiency of the reverse circuit proved to be critical, one

[6]Since links in the core are usually symmetrical, the bandwidth inefficiencies caused by this traffic unbalance also affect packet switching.

could allocate less bandwidth to the return channel for the acknowledgements.

4.4.3 Denial of service

Denial of service is an important concern for TCP Switching. With only a few well-crafted packets one can reserve a huge amount of bandwidth, preventing others from using it. This problem is not new, and it is common to other systems that do resource reservation. Two solutions are possible here: one is to use external economic incentives and penalties to deter a user from taking more resources than he/she needs. The other is to restrict the maximum number of simultaneous flows that an ingress boundary router may accept from a single user.

On the other hand, one of the advantages of TCP Switching is that circuits are reserved exclusively for one flow, so, contrary to packet-switched networks, it is easy to track a circuit back to its source, and it is virtually impossible for others to spoof a circuit or to hijack it without the cooperation of a switch. This inherent authentication makes the enforcement of policies across domains easier than in the current Internet.

4.5 Conclusions and summary of contributions

This chapter has focused on how the existing IP infrastructure can incorporate fast, simple (and perhaps optical) circuit switches. Several approaches to this already exist, but I have proposed a technique called TCP Switching in which each application flow (be an individual TCP connection or other types of flows) triggers its own end-to-end circuit creation across a circuit-switched core. Based on IP Switching, TCP Switching incorporates modified circuit switches that use existing IP routing protocols to establish circuits. Routing occurs hop by hop, and circuit maintenance uses soft state; i.e., it is removed through an inactivity timeout. TCP Switching exploits the fact that our usage of the Internet is very connection-oriented and has end-to-end reliability to provide lightweight signaling mechanisms for circuit management. Finally, this chapter has showed how despite the fine granularity of its circuits, TCP

Switching is implementable with simple hardware support, and so it is capable of providing the advantages of circuit switching to the Internet. TCP Switching is an extreme approach that shows how one can integrate circuit switching in the core of the existing Internet while extracting the benefits of circuit switching that were listed in Chapters 1, 2 and 3: higher switching capacity, robustness, simple QoS and end-user response time similar to that of packet switching.