ephemeral keys. The TLS protocol also establishes a short-term link key when communicating between ORs. Short-term keys are rotated periodically and independently, to limit the impact of key compromise.

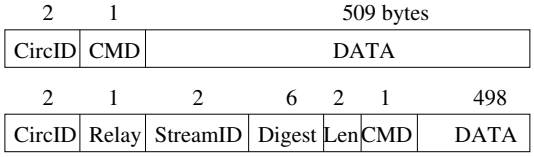
Section 4.1 presents the fixed-size *cells* that are the unit of communication in Tor. We describe in Section 4.2 how circuits are built, extended, truncated, and destroyed. Section 4.3 describes how TCP streams are routed through the network. We address integrity checking in Section 4.4, and resource limiting in Section 4.5. Finally, Section 4.6 talks about congestion control and fairness issues.

4.1 Cells

Onion routers communicate with one another, and with users' OPs, via TLS connections with ephemeral keys. Using TLS conceals the data on the connection with perfect forward secrecy, and prevents an attacker from modifying data on the wire or impersonating an OR.

Traffic passes along these connections in fixed-size cells. Each cell is 512 bytes, and consists of a header and a payload. The header includes a circuit identifier (circID) that specifies which circuit the cell refers to (many circuits can be multiplexed over the single TLS connection), and a command to describe what to do with the cell's payload. (Circuit identifiers are connection-specific: each circuit has a different circID on each OP/OR or OR/OR connection it traverses.) Based on their command, cells are either *control* cells, which are always interpreted by the node that receives them, or *relay* cells, which carry end-to-end stream data. The control cell commands are: *padding* (currently used for keepalive, but also usable for link padding); *create* or *created* (used to set up a new circuit); and *destroy* (to tear down a circuit).

Relay cells have an additional header (the relay header) at the front of the payload, containing a streamID (stream identifier: many streams can be multiplexed over a circuit); an end-to-end checksum for integrity checking; the length of the relay payload; and a relay command. The entire contents of the relay header and the relay cell payload are encrypted or decrypted together as the relay cell moves along the circuit, using the 128-bit AES cipher in counter mode to generate a cipher stream. The relay commands are: *relay data* (for data flowing down the stream), *relay begin* (to open a stream), *relay end* (to close a stream cleanly), *relay teardown* (to close a broken stream), *relay connected* (to notify the OP that a relay begin has succeeded), *relay extend* and *relay extended* (to extend the circuit by a hop, and to acknowledge), *relay truncate* and *relay truncated* (to tear down only part of the circuit, and to acknowledge), *relay sendme* (used for congestion control), and *relay drop* (used to implement long-range dummies). We give a visual overview of cell structure plus the details of relay cell structure, and then describe each of these cell types and commands in more detail below.



4.2 Circuits and streams

Onion Routing originally built one circuit for each TCP stream. Because building a circuit can take several tenths of a second (due to public-key cryptography and network latency), this design imposed high costs on applications like web browsing that open many TCP streams.

In Tor, each circuit can be shared by many TCP streams. To avoid delays, users construct circuits preemptively. To limit linkability among their streams, users' OPs build a new circuit periodically if the previous ones have been used, and expire old used circuits that no longer have any open streams. OPs consider rotating to a new circuit once a minute: thus even heavy users spend negligible time building circuits, but a limited number of requests can be linked to each other through a given exit node. Also, because circuits are built in the background, OPs can recover from failed circuit creation without harming user experience.

Figure 1: Alice builds a two-hop circuit and begins fetching a web page.

Constructing a circuit

A user's OP constructs circuits incrementally, negotiating a symmetric key with each OR on the circuit, one hop at a time. To begin creating a new circuit, the OP (call her Alice) sends a *create* cell to the first node in her chosen path (call him Bob). (She chooses a new circID C_{AB} not currently used on the connection from her to Bob.) The *create* cell's payload contains the first half of the Diffie-Hellman handshake (g^x), encrypted to the onion key of Bob. Bob responds with a *created* cell containing g^y along with a hash of the negotiated key $K = g^{xy}$.

Once the circuit has been established, Alice and Bob can send one another relay cells encrypted with the negotiated