

Staying Alive

High Performance HTTP via Keep- Alives

Alex Kodat
Sirius Software Inc.



HTTP – HyperText Transfer Protocol

- Protocol layered on top of TCP/IP
- Simple request/response protocol
 - Client connects to server
 - Client sends request to server
 - Server sends response to client
 - Client (or server) closes connection
- The protocol used by web browsers
- The protocol used for Web Services (SOAP)
 - Application to application communication
- Protocol used for many other protocols (WAP for example)



Strengths of HTTP Protocol

- Simplicity
- Agnostic about content
 - Despite its name
- Statelessness
 - Server friendly – server doesn't need to maintain state
 - User friendly – Easy to do many things at once and pick up where one left off
- Connectionless
 - Server friendly – possible to support huge numbers of users with relatively small server(s)
 - Server farms possible

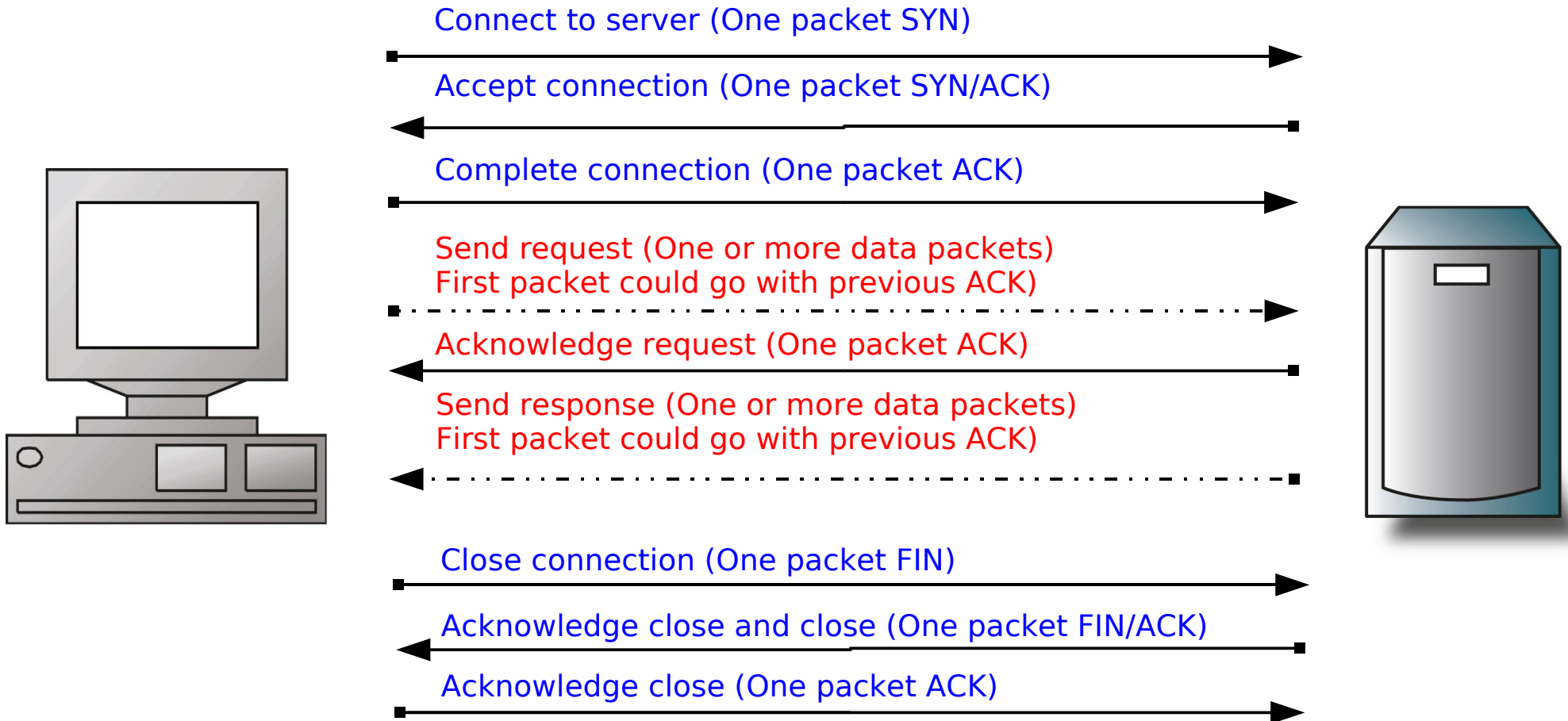


Weaknesses of HTTP Protocol

- Verbosity
 - Only an issue for very small requests
- Statelessness
 - Some applications are necessarily stateful (to various degrees)
 - Can be harder to write conversational applications
- Connectionless
 - So every request requires a new connection
 - Latency city
 - Latency: The time nothing “useful” is happening



An HTTP Request – IP Traffic



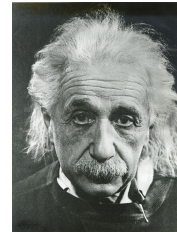
HTTP Request – IP Traffic Score

- Total number of data packets – 2 or more
 - Typical data packet size is 1500 or 4000
 - So most simple requests would be two data packets
- Total number of data control packets – 5 to 7
 - Depending on cleverness and timing of TCP stack and application
 - Data control packets typically small – 40 bytes or so
- Total number of synchronous *turn-arounds* – 3
 - Although, if application clever, third turn-around could be dealt with purely by TCP stack



So What's the Real Cost of All Those Extra Packets?

- Extra network bandwidth
 - At worst case 7 40-byte packets, this is 280 bytes per request
 - But even on a “slow” 10-Mbit ethernet, this adds a mere 0.35 millisecond delay (assuming you can use around 6-Mbits of 10-Mbits)
 - So who cares?
- But synchronous network turn-arounds are typically at least 0.5 milliseconds
 - Distance is the enemy
 - Speed of light is the ultimate limit
 - Turn-around to a server 500 miles away is guaranteed to be at least $1000/186,000$ seconds or 5.3 milliseconds



So What's the Real Cost of All Those Extra Packets?

- In many applications, even on a local network, turn-around time tends to dominate HTTP request response time
- So the synchronous open possibly doubles request response time
- If using SSL even worse, because an SSL handshake requires an extra two turn-arounds (at least)

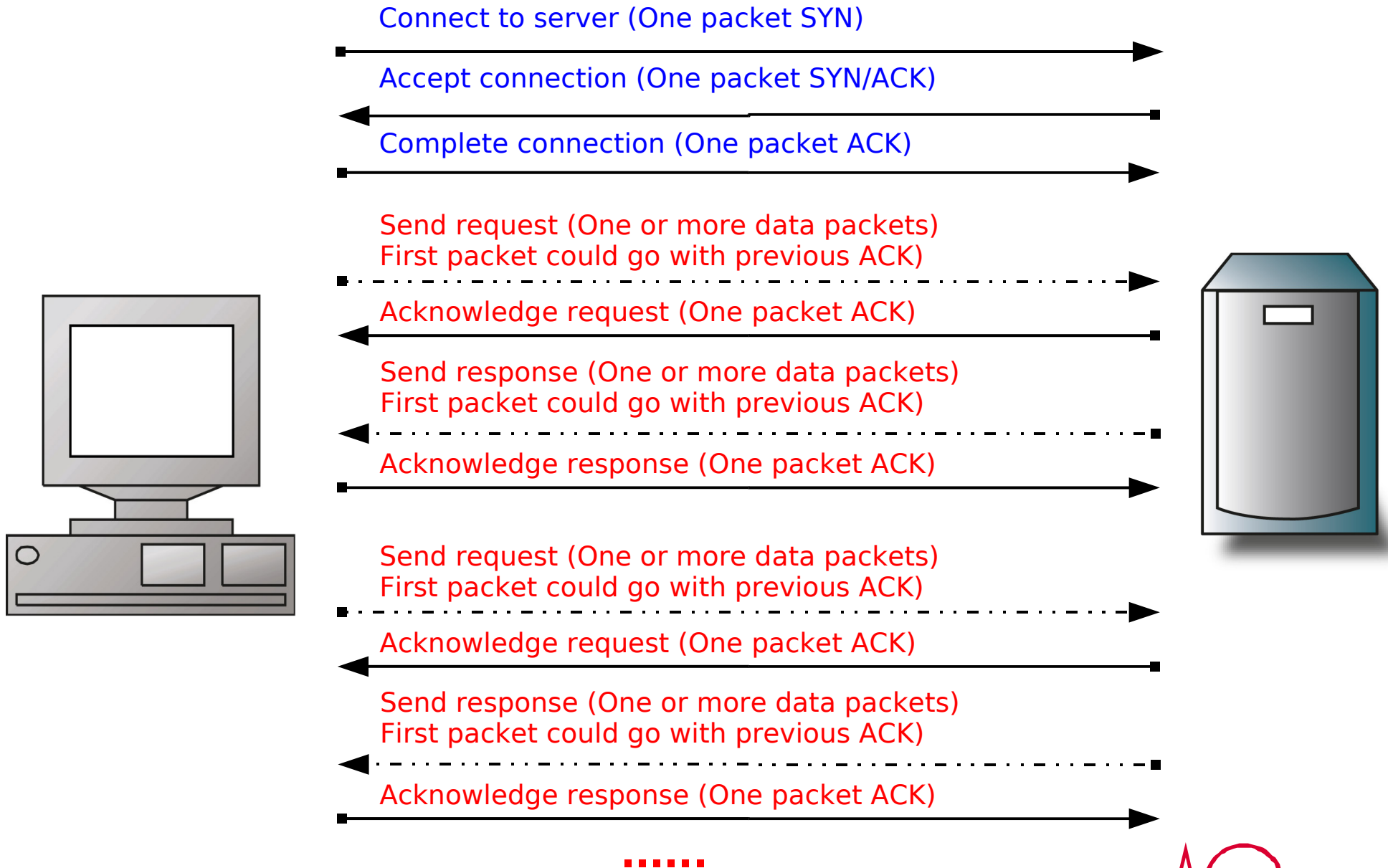


So Suppose We Kept HTTP Connections Open?

- Initial connection handshake would be eliminated for all but first request
- Close handshake eliminated for all but last request
- Request response time should be cut about in half



An HTTP Request – IP Traffic



Keeping Connections Open with HTTP

- Reduces number of data control packets per request to two or zero
- Reduces number of synchronous turn-arounds to one
 - The absolute minimum possible



A Word About Large HTTP Requests and Responses

- Requests and responses bigger than network packet size are broken down into multiple network packets
- Receipt of each part of message must be acknowledged
- But multiple packets are sent before oldest packet acknowledged
 - Up to some negotiated “window” size (typically around 30,000 bytes)
 - So maximum transfer rates ultimately determined by distance and window size
 - For 12,000 mile distance and 30,000 byte window, max rate is 232,500 bytes/second
 - Math left as exercise for reader

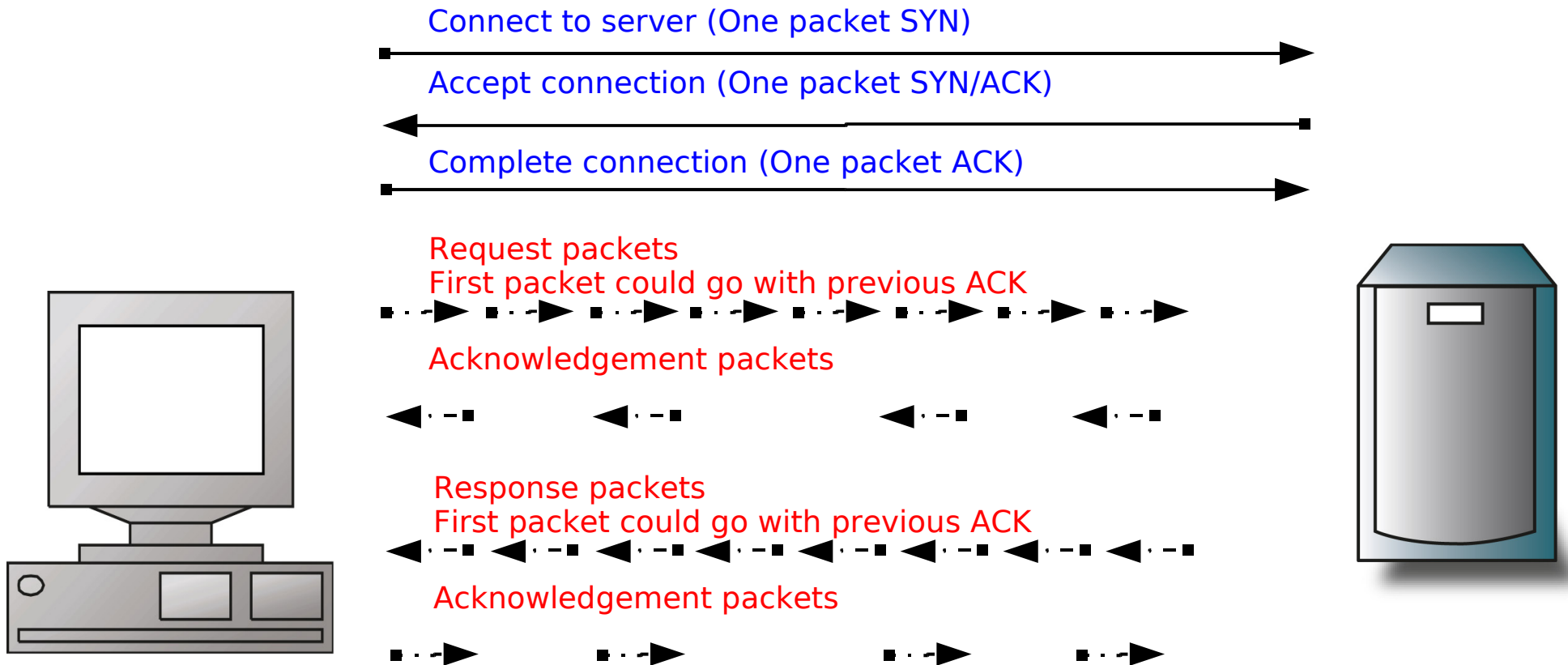


Turnaround Time for Large HTTP Requests/Responses

- Because of TCP window, no extra turnaround time up to window size
- Above window size, extra turnaround time proportional to size of message
 - For message twice as big as TCP window, time required is two turn-arounds
- Since TCP window is typically around 30K, this is only a factor for very large messages



Large HTTP Request/Response – IP Traffic



So Why Not Just Keep HTTP Connections Open All the Time?

- Resources tied down on both server and client for open connections
- On server side, an architectural limit of 64K connections to a port
- Benefits of keeping infrequently used connections open are minor



The HTTP Compromise – Keep-Alive Time

- HTTP connections held open for a set amount of time
 - Minimum of server and client keep-alive time
- After each request keep-alive timer restarted
- Client must be prepared to retransmit a request if connection times out just as request is sent
- So for busy connections, connection held open indefinitely
- Idle connections are closed reasonably quickly
 - Keep-alive timeouts should be in seconds, not minutes or hours



Setting Keep-Alive Timeout for Janus Web Server Ports

- KEEPALIVE parameter on JANUS DEFINE for port
 - Followed by number of seconds to keep connections alive
 - Start with a number like 10
- Might require increasing the number of connections allowed for a port
 - Not recommended for low thread count (10) licensees
- Does **not** require increasing number of users/servers
 - User/server thread not used while connection being “kept alive”
- Requires **no** application changes
- Requires Sirius Mods 6.8 or later



Benefits of Setting KEEPALIVE on Janus port definitions

- Less TCP/IP address space overhead
- Less Model 204 (Janus PST) overhead
 - Especially if using SSL
- Spunkier response time for browsers
 - Web requests to a server tend to be *bursty* – an HTML page often contains references to other content on the same server
 - CSS, frames, images, javascript, etc..
 - So often, a request to a server is immediately followed by several more
 - Being able to reuse existing connection makes these faster
 - If requests usually just HTML then possibly little benefit to keep-alives



Keep-Alive and Proxies

- Whether a proxy chooses to use existing connections for requests from different clients is up to the proxy
 - Janus Web can handle it if it does
- Should a proxy choose to do so, and should all/most requests to Janus Web go through a single or handful of proxies, then huge benefits possible by using keepalives



Keep-Alives and Statelessness

- Use of keep-alives does not affect statelessness of web applications one iota
 - Only the TCP connection is held open, no user/HTTP data is maintained
- However, keep-alives can benefit stateful apps by holding the connection open
 - Will work in conjunction with Janus Web Legacy support or janus Web persistent session support (`$web_form_done`)



What About Web Services?

- Web services always use HTTP
- Because web services invoked by applications, often several web services invoked to the same server in a single request
- So potentially big benefits to using keep-alives for web services
 - Especially for server to server apps where two or a handful of servers are constantly exchanging web service requests
- KEEPALIVE for Janus Web Server ports works fine for web services provided by that port
- Web services clients require KEEPALIVE on client port definition



Setting Keep-Alive Timeout for Janus Client Ports

- KEEPALIVE parameter on JANUS DEFINE for (CLSOCK) port
 - Followed by number of seconds to keep connections alive
 - Start with a number like 10
- Might require increasing the number of connections allowed for a port
 - Though not if only communicating with a handful of servers
- Only used by HTTP helper objects
 - No harm (or benefit) to other client socket applications
- Requires Sirius Mods 6.8 or later



Use of Keep-Alives with HTTP Helper

- When Post or Get is done, Janus looks for open (kept-alive) connection
- If open connection exists, it is reused
- If no open connection exists (or all open connections busy) a new connection is initiated
- After request completes, connection placed into keep-alive state (held open) for KEEPALIVE seconds
- If connection not reused after KEEPALIVE seconds, it is closed
- No application changes are necessary

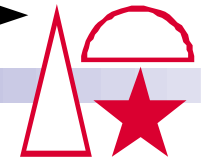
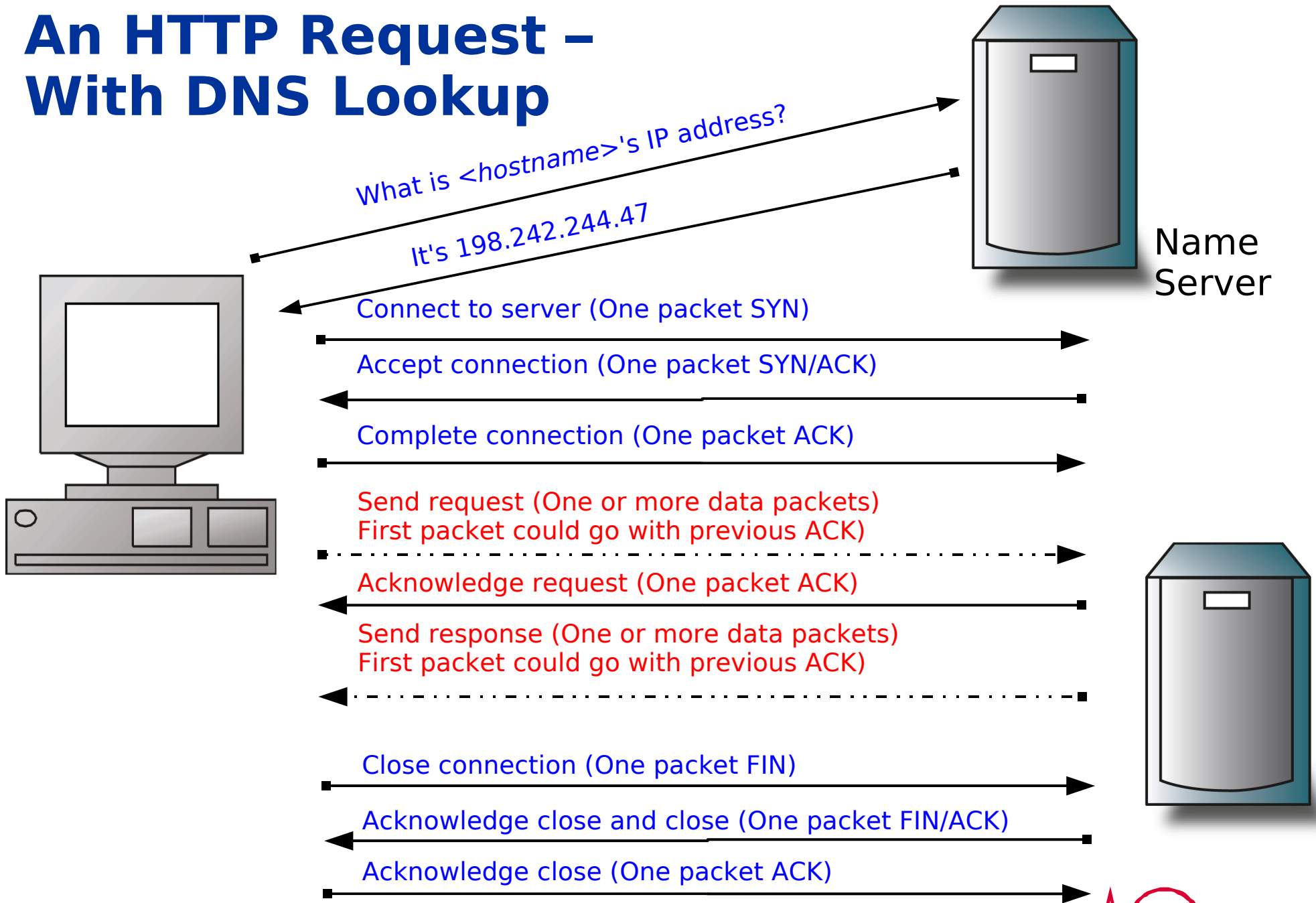


Name Server Lookups

- Client requests **always** to an IP address
- But IP addresses change frequently and can be hard to remember
- So host names are typically used in most applications
- Host name must be mapped to an IP address before connecting to server
- This is done with a name server lookup



An HTTP Request - With DNS Lookup



DNS Lookups with HTTP

- Adds another turn-around to an HTTP request
 - Though to a different machine – the name server
 - Name server might (should) be on the local network
- Ideally, the name server is on the same host as the client
 - But this can be a hassle
 - Name servers need some care and feeding
 - If client is on mainframe, mainframe name servers provide by IBM not ideal
 - Use caching only name servers



Local DNS Caching for Janus Sockets

- Indicated by CACHE parameter followed by maximum number of entries to cache on JANUS NAMESERVER command
 - Length of time entries valid indicated by MAXTTL parameter
- Available with Sirius Mods 6.8 and later
 - JANUS NAMESERVER command now also supports multiple nameservers (for redundancy)
- Initial lookup for name goes to nameserver
- Subsequent lookups for same name found in local cache
 - Cache maintained on LRU basis
 - Virtually no overhead to lookups



Web Services Tuned to the Max

- Local DNS caching on client
- KEEPALIVEs for the HTTP connections
- Means **no** non-request related overhead



Monitoring Keep-Alive/DNS Caching Performance

- JANUS TSTATUS shows number of connections and number of reused connections by port
- New SirMon system states to track DNS caching success
 - DNSRTOT and DNSRCACH of particular interest
- Audit message indicate whether new connection used or existing connection reused



Conclusions

- New facilities for keep-alives and DNS caching in Sirius Mods 6.8 can improve application performance
 - Especially beneficial for web services
- Requires no application changes
- So give it a spin



You Can Tell By the Way I Use My Server, I'm a Geeky man, No Time to Talk....

Any questions?



Note: My apologies to any songs you might have stuck in your head
the rest of the day



Sirius Software, Inc.